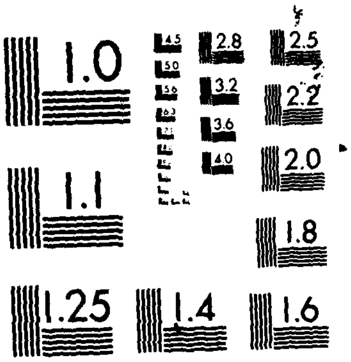
AD-A117 995 SOFTWARE ENGINEERING SEMINAR (7TH) 28 APRIL - 2 MAY 1980 ELGIN AIR FORCE BASE FLORIDA(U) RANGE COMMANDERS COUNCIL WHITE SANDS MISSILE RANGE NM DATA RE.

1/2

UNCLASSIFIED 02 FEB 82

F/G 9/2

NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DR&CG

7TH SEMINAR

SOFTWARE ENGINEERING

2 FEBRUARY 1982

HEADQUARTERS TACTICAL FIGHTER WEAPONS CENTER
NELLIS AIR FORCE BASE, NEVADA

RANGE COMMANDERS COUNCIL

WHITE SANDS MISSILE RANGE
KWAJALEIN MISSILE RANGE
YUMA PROVING GROUND

PACIFIC MISSILE TEST CENTER
NAVAL WEAPONS CENTER
ATLANTIC FLEET WEAPONS TRAINING FACILITY
NAVAL AIR TEST CENTER

EASTERN SPACE AND MISSILE CENTER
ARMAMENT DIVISION
WESTERN SPACE AND MISSILE CENTER
AIR FORCE SATELLITE CONTROL FACILITY
AIR FORCE FLIGHT TEST CENTER
AIR FORCE TACTICAL FIGHTER WEAPONS CENTER

DATA REDUCTION AND COMPUTING GROUP
RANGE COMMANDERS COUNCIL

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

151

7TH DR&CG SEMINAR

SOFTWARE ENGINEERING

DATA REDUCTION AND COMPUTING GROUP
RANGE COMMANDERS COUNCIL

2 FEBRUARY 1982

HEADQUARTERS TACTICAL FIGHTER WEAPONS CENTER
Nellis Air Force Base, Nevada

Published by

Secretariat
Range Commanders Council
White Sands Missile Range,
New Mexico 88002

DISCLAIMER

This document has been published for information purposes only. The material contained herein does not necessarily represent the position conclusions of the Range Commanders Council (RCC). This document has been reviewed for OPSEC considerations, distribution unlimited.

TABLE OF CONTENTS

LIST OF ATTENDEES

NAME	ADDRESS
Thomas Berard	AFFTC, Edwards AFB, CA
Don Goodall	White Sands Missile Range, NM
Eugene H. Dirk	White Sands Missile Range, NM
Theron E. Anders	Kentron, Huntsville, AL
Linus V. Kovar	CSC, Vandenberg AFB, CA
Gary L. Reed	CSC, Vandenberg AFB, CA
Hurbert M. Horton	New Technology Inc Huntsville AL
Avery Baswell, Jr.	KMR, Huntsville, AL
Richard I. Mitchell	BMD, Nellis AFB, NV
David Y. Hsu	CDEC, Ft. Hunter Liggett, CA
Joseph M. Weinstein	CDEC, Ft. Hunter Liggett, CA
John J. Reisig	CDEC, Ft. Hunter Liggett, CA
Herbert L. Setzler	475 Test Sq. Tyndall AFB, FL
James E. Johnson	NATC, Patuxent River, MD
Robert N. Barry	AFFTEC, Edwards AFB, CA
Thomas E. Ford	PMTC, Point Mugu, CA
Ken Mulkey	554 RG/XD, Nellis AFB, NV
William J. Egan	PMTC, Point Mugu, CA
Stacy Lynn	VEDA, Inc., Nellis AFB, NV
David B. Culp	BMD/EWCAS, Nellis AFB, NV
James A. Papa	AFFTC, Edwards AFB, CA

LIST OF ATTENDEES (con.)

NAME	ADDRESS
W.J. Kirklin	ESMC, Patrick AFB, FL
William Swartz	Sandia National Labs, Albuquerque, NM
John Greenwald	PMTC, Point Mugu, CA
J.R. Gush	554 RG/DOXS, Nellis AFB, NV
Wendell D. Thomas	PMTC, Point Mugu, CA
Jose Rodriguez, Jr.	AFWTF, Rosevelt Roads, PR
J. Paul Welch	AD/KRA, Eglin AFB, FL
Gwendolyn E. Hunt	PMTC, Point Mugu, CA
William Reich	PMTC, Point Mugu, CA
Larry Hackamack	YPG, Yuma, AZ
Edward Hinton	Logicon, Vandenberg AFB, CA
William Chrisman	554 RC/DOXS, Nellis AFB, NV
Lyle Majors	554 RG/XD, Nellis AFB, NV
F.J. Bailey, III	ESMC, Patrick AFB, FL
James R.S. Manion	NWC, China Lake, CA

OFFICIAL AGENDA

DATA REDUCTION AND COMPUTING GROUP
OF THE
RANGE CODMMANDERS COUNCIL
SOFTWARE ENGINEERING SEMINAR

Tuesday - 2 February 1982

0830 - Registration

0900 - Opening Remarks - Walter H. Nolin, Hq ESMC, Chairman

0910 - Software Engineering Series in the Federal Government -
Gwendolyn E. Hunt, PMTC, Pt Mugu, CA

1000 - A Near Real-Time System for the Reduction/Production of
Cinetheodolite Data - James R.S. Manion, NWC, China Lake, CA

1100 - Software Maintenance - Coming Out of the Closet -
W.J. Egan, PMTC, Pt. Mugu, CA

1200 - 1330 - LUNCH

1330 - Success Through Composite Structured Design -
W.J. Kirklin, ESMC, Patrick AFB, FL

1430 - Software Acquisition Within a System Acquisition -
John Greenwald, PMTC, Pt. Mugu, CA

1530 - Flight Test Oriented Precompiler System (FLTOPS) -
Thomas Berard, AFFTC, Edwards AFB, CA.

1630 - Closing Remarks - Walter H. Nolin, Chairman

1635 - Adjourn

PREFACE

The Software Engineering Seminar was conceived at the 53rd Meeting of the Data Reduction and Computing Group (DR&CG) which was held 28 April through 2 May 1980 at Eglin Air Force Base, Florida.

It was agreed by the DR&CG membership, since many member ranges and centers were committing an increasing amount of resources to design, develop and implement software systems, that a seminar to discuss and disseminate technical information would be most beneficial. This vital area is demanding more and better management visibility as the cost of software systems to total systems cost is increasing at an alarming rate. The Executive Committee (EC) of the Range Commanders Council (RCC) concurred with the proposed Software Engineering Seminar. Our target date was for the 56th meeting of the DR&CG to be held in February 1982 at the Air Force Tactical Fighter Weapons Center (AFFTC), located at Nellis Air Force Base, Nevada.

Software development is big money, even considering only defense systems. It has been estimated that the Department of Defense will spend nearly 10 billion dollars per year over the next 5 years on computer software alone. Overall productivity must be increased during design, development and implementation of software systems. If overall productivity of software development can be increased by only 1 percent, this could result in an annual savings of approximately 100 million dollars a year.

Software development costs are predominately people costs. If one can increase the people productivity, then management has made a significant contribution to controlling the rising costs. Two prime considerations in achieving increased productivity are automation and education. It seems that more monies can be wisely spent in these two areas. What are the alternatives? The alternatives are clearly unaffordable software and inefficiently utilized computer hardware.

The primary objective of this seminar was to share information, techniques and methods of Software Engineering in an effort to increase productivity.

The seminar provided an outstanding opportunity for attendees: (1) to meet and associate with fellow mathematicians, engineers, and computer scientists from government and industry, (2) to exchange ideas, information and experiences related to software engineering, and (3) to aid in the formulation of workable solutions to some of the complex problems confronting management in this area.

We hope that these papers communicate some of the interest stimulated by topics that were presented orally at the actual seminar.

Walter H. Nolin
Computer Services Division
Patrick Air Force Base, FL.

SOFTWARE ENGINEERING SERIES IN THE FEDERAL GOVERNMENT

PRESENTOR: GWENDOLYN E. HUNT
ASSOCIATE, SYSTEMS TECHNOLOGY
OFFICE, PACIFIC MISSILE TEST CENTER

ABSTRACT

SOFTWARE ENGINEERING PROJECT
PROFESSIONAL COUNCIL OF FEDERAL SCIENTIST
AND ENGINEERS

The West Coast Region of Professional Council of Federal Scientists and Engineers has been exploring the difficulty encountered by the lack of specific classification and qualification standards in the Federal Government for positions in the emerging field of software engineering. This paper provides an account of the efforts to develop the software engineering occupational series. It also describes interim approaches by the Department of Navy, and some of its field activities, to alleviate the problems of recruitment and retention, undefined career paths and salary inconsistencies.

Finally, a prognosis is provided of the success of the project, together with the new initiatives which are being studied.

SOFTWARE ENGINEERING SERIES IN THE FEDERAL GOVERNMENT

PROFESSIONAL COUNCIL OF FEDERAL SCIENTISTS AND ENGINEERS
ADVISORS TO THE DIRECTOR, SAN FRANCISCO REGIONAL OFFICE
FRANCIS V. YANAK, DIRECTOR

BACKGROUND

In the mid-sixties, the Department of Defense (DOD) began its trend toward the acquisition of weapon systems embodying digital computers as a primary component of a total system.

These computers performed the functions of data storage, process control, and complex decision making. In order to perform these functions, the computers required a set of instructions and data which was narrowly called software. The continued miniaturization of computer hardware through gigantic technological advances forced an acceleration of this trend in the seventies. Simultaneously, decreased national productivity, an unstable economy, energy shortfalls, and the increase in federally supported social services fostered the proliferation of computers in the non-defense sectors of the Federal Government.

In both environs, the complexity of the development, operation, support and management of these computer-based systems spawned a community of government employees whose required knowledges, skills and abilities (KSA's) transcended traditional academic disciplines such as mathematics, engineering, physics, and the like. These required KSA's have been described in subject matter literature since the late sixties as "Software Engineering," with the unique individual processing these KSA's being known as a "Software Engineer." A recent software engineering text provides the following:

"Among the many definitions of software engineering proposed since 1970, the most accurate and descriptive was by F.L. Bauer of the Technical University, Munich, Germany, in 1972. His definition can be stated:

> The establishment and use of sound Engineering Principals (methods) in order to obtain economically software that is reliable and works on real machines

This definition of software engineering encompasses the keywords that are the heart of all engineering discipline definitions: sound engineering principles, economical, reliable, and functional (works on real machines)." (2:9)

To paraphrase, software engineering is the application of knowledge of mathematical and physical sciences acquired by special education, training and experience to the various aspects of software system design, development, and management essential to ensure effective, efficient and economic utilization of computer system resources.

The software engineer is responsible for various aspects of software system design, development, and management essential to ensure effective utilization of computer system resources as elements of major physical or environmental systems which incorporate one or more specific engineering disciplines. The computer systems are generally embedded and/or integrated within a major system complex and provide direct real-time control of and/or perform specific tasks within one or more of the system functional elements.

While there is an equally important required skill of understanding the computer and its languages, the software engineer must understand the operation, functions, and interfaces of the total system in order to effectively solve complex problems necessary to design algorithms and instructions for the computer, resulting in an economical and reliable system.

Private industry has already recognized this emerging technological area and has established a career field for the software engineer. With the demand far exceeding the supply, industry is also offering premium salaries to applicants. Without a designated software discipline, the federal service lags behind industry and this compounds the recruiting problems. The software engineering job category must be recognized in the public sector in order to overcome the recruitment and retention obstacles which already include salary disparity within the industry and lack of specific, well-defined career patterns in this field.

The West Coast Regional Council of Professional Scientists and Engineers had for some time been exploring the difficulty caused by the lack of appropriate classification and qualification standards in the Federal Government. The Council is composed of senior civilian representatives of the DOD and other federal civilian activities in the western region which employ 50 or more professional employees engaged in research, development, test and evaluation. It was established as an advisory group to the San Francisco Regional Office of Personnel Management (OPM) by the director, Francis V. Yanak. Through its involvement with the many initiatives to upgrade the quality of the federal technical work force, it was successful in obtaining special pay rates for engineers; first in the western region, then nationwide. A more recent effort is to obtain approval to extend this special pay status to all scientists.

The software engineering project was officially intiatiated in 1979. The approach was that the Western Regional Office, utilizing field activity resources accessible through its Professional Council of

Scientists and Engineers, undertake the development of a new engineering series for Software Engineers. The first phase of the effort was to conduct an occupational survey resulting in a definition of a new series. Later efforts would be concerned with full classification and qualification standards under the guidance of the Standards Development Center OPM, Washington, DC.

At the same time, the Department of the Navy System Commands, Material Command and Field Activities were being plagued by the same problem. The Master Plan for Tactical Embedded Computer Resources, a Naval Material Command document states.

> "Weapon systems software acquisition and maintenance are becoming sufficiently important that consideration should be given toward establishing a separate career field for weapon system software engineers. Also, software acquisition and maintenance is sufficiently complex and challenging so that career incentives should be developed to attract and retain good personnel." (4:3-39)

The document suggests an action program which would:

- Identify the software personnel requirements.
- Review the current classification procedures.
- Review training requirements and training capabilities for preparing software engineers.
- Recognize this expertise as critical and develop necessary career programs and incentives.

Separate and independent initiatives began sprouting throughout the government as more and more dependency upon the computer was evidenced.

The Air Force added software engineering courses to the curriculum at the Air Force Institute of Technology (AFIT) and the Naval Postgraduate School offered a degree in Computer Science.

The prime thrust of all this activity, however, was a reaction to the computer technological explosion and many efforts proceeded redundantly rather than synergistically.

MAJOR ISSUES

The problems of providing sufficient numbers of qualified software personnel did not lend themselves to short-term management resolutions. The full impact on qualification requirements, and hence on recruitment and placement, from the establishment of a highly specialized series such as software engineering had to be thoroughly analyzed to assure that the benefits to be gained outweighed the disadvantages to be incurred. The vast number of federal employees already performing duties requiring specific KSA's of computer resources and classified to other occupational series, some non-professional such as the Computer Specialist, caused the following issues to be considered:

1. Is Software Engineering a "professional" occupation;i.e., requiring a engineering academic preparation. According to the position classifiction standards for the Engineering series:

> "Whether an occupation is placed in the engineering group of physical sciences group depends upon whether the nature of the work and the qualifications required for its performance are predominantly identified with an engineering or physical science discipline. It is the common core of professional knowledges and abilities representing a discipline required for performance of the work which distinguishes series within occupational groups. Areas of application or investigation are of secondary significant. . . .
>
> In some cases, where large numbers of multi-discipline positions constitute what may be considered to be a new profession or occupation with a common core of duties and qualifications required, a new series is established, e.g., Soil Conservation Series." (5:11,12)

By definition, the software engineer is responsible for the application of engineering principles, theory and concepts to the development of the software which works, reliably and economically. These requirements imply a specific academic progression to a certified level of competence, even though many years of experience may sometimes narrowly suffice. For example, an individual who "engineers" weapon system software must have knowledge of the avionics subsystem within the weapon system if he is to develop, design, or maintain the embedded weapon software and/or firmware. One must understand the environment in which the weapon system is employed, i.e., threats, countermeasures, etc. In addition, there is the equally important skill of understanding the computer and languages. Both these skills are required to be effective in designing algorithms and the instructions for the weapon system embedded digital computer. This example holds true for large, complex, non-embedded systems which control many subsystems as an integrated entity.

2. What is the impact of the lack of series definition, specialization, classification and grade criteria?

Since there are no classification standards in federal service for this engineering field, federal agencies have been forced to establish and fill professional software positions from related disciplines such as electronic engineers, mathematicians, computer specialists, aeronautical engineers and general engineers. This clouds the career patterns of the selectee, since the individual is not "set apart" in title and/or series as to his specific expertise in software engineering. The lack of series definition and specific classification criteria create an opportunity for misunderstanding in grading positions since the classifier must search for an appropriate standard or standards which will properly grade the duties of the software engineer.

3. Are there recruitment and retention obstacles?

Industry has recognized the need for a career field for the software engineer. Given the fact that demand for software engineers far exceeds the supply, and that premium salaries are being offered by industry, the federal government is lagging behind in recruitment and retention. At the entry level, salaries offered by private industry exceed federal salaries by as much as $10,000 or more.

Once in the federal service, the lack of a defined career path for software engineers impacts upon retention.

Although the government provides the opportunity for challenging work, and the opportunities for meaningful advancement, competitive salaries are necessary in order to attract and retain highly qualified personnel in this new and emerging discipline.

4. What is the impact on employees with other series designations who might be subject to reclassification?

Positions currently properly classified will not be impacted. Those classified to a professional engineering or mathematics discipline and whose primary duties are software engineering would require a change to the new engineering series. Those properly classified in the non-professional GS-334 series will remain as they are.

The qualification standards must be developed such that the basic requirements and alternate requirements permit the placement of all incumbents of "certified" software engineering positions.

These issues were given primary consideration; and, after many ad hoc discussions, separate attempts at alleviating the problem within specified time windows were initiated. In the following paragraphs, an account of representative efforts is discussed.

DEPARTMENT OF NAVY EFFORT

The Navy's investigation of the problem began with 20 Navy representatives attending a meeting in August of 1979 to study the computer software engineering field in order to better understand this emerging occupation and the role it plays within the Automatic Data Processing (ADP) community.

The Navy's plan of action included:

1. Define the Scope of the Study

   - *Develop an overall plan for the study including*

     - Study format
     - Methodology
     - Milestones

   - Tie-in with the OPM - Professional Council for Federal Scientists and Engineers, San Francisco Region, Project for a Software Engineer Classification standard

   - Identify major functional areas

   - *Identify major problem areas and issues*

2. Identify Procedures and Assignments

   - Define workload and assignments
   - Conduct fact-finding
   - Develop a draft of findings
   - Coordinate with other offices and activities in geographical area/systems command
   - Corroborative fact-finding by OP-141C1

3. Develop Final Draft of the Study

   - Review and develop draft Interpretive Memorandum
   - Circulate draft for review and comments
   - Complete final draft for publication

4. Publish Interpretive Memorandum on Computer Software Engineering Positions.

Some resistance to the study effort was evident by those activities with a large number of software personnel classified under the 334 standard: however, the majority of the new activities agreed that a special classification series should be established for the long term.

Many problem areas and issues were exposed, the most pervasive issue was an uncontested description of the software engineering discipline and the associated minimum basic qualifications.

The final result of this effort was an Interpretive Memorandum to aid in the classification process. (3)

INTERPRETIVE MEMORANDUM

The Interpretive Memorandum which was formally issued in [illegible] 1981 from Navy Personnel Headquarters, offers intensive classification guidance on software engineering positions which was completed at Navy Personnel Headquarters. It provides criteria for determining the classification of engineer and other professional positions involved with computer software engineering for embedded or similar computer systems. "Software engineering," as defined in this memorandum, covers the engineering work pertaining to the research, development, design, testing, production, installation, maintenance, operation and other functions relating to computer programs and the data required to allow the computer to perform its functions. This guide provides occupational information and grade level criteria for the classification of Navy positions requiring the performance of professional technical work in the field of computer software engineering.

NAVAL SURFACE WEAPONS CENTER (NSWC)

In the Naval Surface Weapons Center opinion, the Interpretive Memorandum did not solve the problem. The Naval Surface Weapons Center has been concerned with identifying the academic preparation and the actual preparation of personnel. (Reference appendix A.) Because of the different backgrounds of those involved in the study, an early decision was made to ignore the titles of positions and the job descriptions. NSWC began by identifying the knowledge areas important to the development of the Navy systems for which NSWC has, or could have, responsibility. These knowledge areas, considered to be necessary for those individuals developing successful systems consist of:

1. Controls - controls, information feedback systems, basic systems distinctions (open loop, closed loop, hierarchical, etc.).

2. Process exposure/dynamic interrelationships - time-dependent behavior, system interactions, the "process" concept, cross effects, binding time, process communications, cooperation and competition.

3. Design principals - the principals of engineering (or the scientific method), elements of the design activity (specification, analysis, decomposition, synthesis, testing), maintenance and reliability.

4. Interpersonal communication skills - written and verbal communication, team participation and team leadership.

5. Functional capabilities of digital hardware - logical structure and composition. (This area was recognized to be potentially divisible into computer hardware and digital non-computer hardware.)

6. Software design technology - system life cycle, specification techniques (e.g., PSL/PSA, Workbook, Jackson, etc.), development techniques (e.g., chief programmer, structured walk-through, design reviews, builds, code reading), documentation, modification and maintenance.

7. Evaluation - systems analysis techniques, models and modeling identification or creation of alternatives, characterization of tradeoffs.

8. Systems integration - component and subsystem testing, system reliability, progressive testing, diagnostic capability, degraded mode options, recovery.

9. Programming systems techniques - programming languages, systems programs, structured programming, modularity, stubs, program documentation, program testing.

10. Human factors engineering - human/machine interface, dialogue design, prompting, "trainability" and "learnability," adaptability and design and change. (This area was recognized as potentially divisible into software design for human use and hardware design for human use.)

In addition, a plan has been devised by the Software Engineering Committee for the acquisition and training of software engineers for the Naval Surface Weapons Center. It was recommended that all newly hired persons who are intended for organizations involved in software development and do not already have an appropriate background be included in the Software Engineering Development Program. Others who wish to change from their present jobs to developing software will also be included.

The plan assumes that the trainees have no detailed knowledge of computers but that they do have at least a BA degree or equivalent in one of the scientific fields. Training opportunities will also be provided for experienced software development personnel through a series of short courses conducted at the Center.

The plan includes a two-step training program including formal classes as well as on-the-job training (OJT) with specific training experiences to be added as needed. (Reference appendix A.)

Pacific Missile Test Center (PMTC)

At the Pacific Missile Test Center, the impact of the shortage of employees having software engineering expertise is severe. It causes understaffing of current operations and overworking of available people. There is great fluidity in labor supply which leads to round-robin attrition, lost continuity and an unbalanced work force. High attrition leads inevitably to a high risk environment in terms of performance, schedule, and cost.

It became increasingly evident that a plan to overcome this growth problem was needed to increase the supply of skilled software engineers from within the organization. The proposed program addressed a modular approach which

1. offered a career training program at the undergraduate level.

2. provided for career branching after the undergraduate program is completed.

3. offered several gratudate certificate programs to employees currently having a bachelor's degree in the professional fields.

4. provided for updating state-of-the-art to employees currently working in the software field.

The objective of this program would be the potential creation of new resources for the areas of software, electronic warfare , and test and evaluation.

The Career Development Division was tasked to design a Software Engineering Career Development Program which would offer modular training opportunities and attract potential resources to the software engineering function at the Center.

A survey of need for this function was conducted in late fiscal year 1977. The scope was limited to software support for scientific and engineering functions and excluded management, business, and supply functions. The training program was to be designed around requirements for the tactical fighter weapons system's software requirements, since the biggest "gap" was occurring within this function. Further, it was felt that any training program designed to meet needs for this highly specialized area would automatically include the knowledge, skills, and abilities necessary to meet the requirements in the excluded areas.

The survey indicated an immediate requirement for 46 new journey employees to meet current demands; 10 additional requirements for fiscal year 1978; and 11 additional requirements for fiscal year 1979. When coupled with retirements and attrition rates over a 5-year period, it became evident the need for qualified software engineers far exceeded the supply. In addition, many of the current employees badly needed training to keep abreast of the exploding technological changes occurring in the software field.

An intensive recruitment program was then mounted to attact such resources to the Center. In addition to attracting recent college graduates at the entry level, attempts were made to capture intermediate or journey professionals in the field. Although some gains were made, the competition for resources in other government agencies and in private industry created an environment where the attrition rate was higher than the accession rate.

Concurrent with the above efforts to attract employees to the software function at the Center, a major Personnel Evaluation and Audit was conducted by the Office of Personnel Management. Findings of the evaluation team included many high-grade positions recommended for downgrading. As a result, Reduction-in-Force placements occurred, and many of the supervisory/managerial positions in the software function suddenly had new employees who were limited in the knowledge/abilities of the software function.

It was in this climate that the need to design a program aimed at attracting new employees into the field; provide update training to current software employees; provide supervisory/managerial overview orientations; and provide for some type of upward mobility opportunity which would attract and retain employees over a longer period of time in an effort to "grow our own" software employees became paramount.

This effort was separated into two phases:

Phase I - provide for a Task/Competency Needs Assessment. This phase was completed during fiscal year 1978.

Phase II - Design Modular Training which would include all of the elements addressed above. The objective of this phase was to design and implement modular training which would include requisite knowledges/skills to satisfy each element of the overall program. The highest priority in the modular development was the providing of state-of-the-art training to current employees. This was due to the high attrition rate of employees having the requisite skills; the rotation of new supervisors into software functional areas, and the recruitment of personnel having less than adequate knowledge/skills/abilities. This combination presented a serious performance/product problem for the center.

The Phase I product was the identification of knowledges, skills and abilities required of employees in each of the occupations assigned to the tactical fighter weapons [illegible]. These knowledges, skills, or abilities were then translated into the development and presentation of 25 courses which offered [illegible] a state-of-the-art training believed to [illegible] the best [illegible]. Three-fourths of the courses in this module [illegible] to current employees at [illegible] once, and feedback from supervisors indicate they see immediate results in improved performance [illegible] employees.

Appendix B addresses the [illegible] training program; the Phase [illegible] product.

PROFESSIONAL COUNCIL EFFORTS

The intiatives descrited in the preceding paragraphs represent short-term, stop-gap measures and may in themselves solve some of the problems of the organizations involved.

However, a long-term commitment by the public sector mangagement to develop personnel programs which ensure adequate numbers of software personnel for complex system software acquisition and maintenance is needed.

The effort by the Professional Council is considered the first step to that end. In 1979, the Professional Council established a committee to work with OPM in the development of appropriate classification and qualification standards for engineering positions. The Pacific Missile Test Center (PMTC), Point Mugu, member, K. I. Lichti, was designated action officer for the project. His subcommittee consisted of the following persons:

Technical Experts

G. HUNT	Navy, Pacific Missile Test Center	Subcommittee Chairperson
S BERMAN	Navy, Pacific Missile Test Center	Subcommittee Member
G. WROUT	Navy, Pacific Missile Test Center	Subcommittee Member
D. NAURATH	Navy, Pacific Missile Test Center	Subcommittee Member
J. BOK	Navy, Pacific Missile Test Center	Subcommittee Member
H. JOHNS	Navy, Pacific Missile Test Center	Subcommittee Member
J. SALAZAR	Air Force, Vandenberg AFB	Subcommittee Member
D. FARREL	Navy, China Lake	Subcommittee Member
J. HOWELL	Air Force, Edwards Air Force Base	Subcommittee Member

Personnel Analysts

J. BENNISON	Office of Personnel Management WR	Analyst Team Chairperson
D. PIUSER	Navy, Pacific Missile Test Center	Analyst Team Member
V. VERNEUILLE	Air Force, McClellan Air Force Base	Analyst Team Member

The tentative schedule for 1980 was established as follows:

ACTION	DATE	LEAD ACTIVITY
Draft Questionnaire and Series Description	20 June	OPM-Western Region
Establish Action Plan	20 June	PMTC/OPM-WR
Draft Cover Letter	3 July	OPM-WR
Solicit Distribution Lists from Council Subcommittee	3 July	PMTC
Distribution Lists due to PMTC	10 July	Council Subcommittee
Generate Composite Distribution List	11 July	PMTC
Finalize Questionnaire and Series Description	14 July	PMTC
Distribute Questionnaire	15 July	PMTC
Establish Analysts Committee	18 July	PMTC/OPM-WR
Questionnaire Response due to PMTC	1 August	Distribution
Complete Data Search	5 August	PMTC
Complete Initial Data Analysis	8 August	Analyst Committee
Review Analysis	19 August	Council Subcommittee
Develop Series Definition Package	8 September	All
Deliver Package to Council	15 September	PMTC
Review	26 September	Council
Deliver Package to Standards Development Center, OPM, Washington, DC	30 September	Council

The subcommittee distributed a questionnaire to a cross-section of federal activities. The analyst team convened on August 19-20, [illegible], and completed the initial review of responses. Of 285 questionnaires mailed to [illegible] activities in [illegible] departments/agencies, 25[illegible] were returned by the activities. The quality of the input ranged from cursory to very thorough. The enclosures and additional materials furnished by activities such as the Naval Surface Weapons Center, Dahlgren, Virginia, and the Naval Weapons Center, China Lake, California, were indicative of the [illegible] in this topic, and other studies which had already been undertaken. The respondees indicated that approximately [illegible] positions [illegible] fall within coverage of the Software Engineer series (GS-8XX) as defined in the questionnaire. The number of positions identified exclusively with a requirement for an engineering degree was a significantly lesser number than 1200. The most significant population, however, was in the Electronics Engineering Series GS-855. Though some jobs were included in the Computer Specialist series GS-334, the vast majority, estimated at over 95%, were classified to professional series in the GS-800, GS-1300 or GS-1500 groups.

The overwhelming response reflected the viewpoint that the nature of the work to be performed required entry level professional training comparable to that attained in a 4-year degree program leading to a BS in engineering or computer science or mathematics or closely related disciplines. In addition to the discussion of minimum qualifications, a discussion of the required knowledges, skills, and abilities for full performance was provided. The majority viewpoint characterized the occupation as grounded principally in engineering fundamentals and computer knowledges, with related knowledges of mathematics and physical sciences, and abilities in general management and communications.

The questionnaire which asked for distinguishing characteristics of software engineering from other occupations was not addressed by a significant number of respondees even though principal differences were detailed. The team did not summarize inputs regarding the Computer Specialist series since the trend on minimum qualifications clearly characterized the work as "professional."

From the data presented it was the Council's opinion that the software engineering occupation should be established as a fully professional engineering series. A very significant number of 1239 positions identified with the proposed series in the questionnaire responses are currently classified in the Electronic Engineering Series GS-855. This fact substantiates the conclusion that professional engineering training is the over-riding qualification requirement.

The Council further concluded that with approval of the new series, the federal service will be competitive with industry and will significantly benefit in its efforts to attract and retain a fully qualified staff to meet the software requirements of the Federal Government.

Appendix C contains excerpts from the report as sent to OPM, Washington, DC, on 30 September 1980.

STATUS AND PROGNOSIS

The current state of the individual short-term efforts appear somewhat encouraging while attainment of the long-term objective looks dismal. The reports on these individual efforts are:

Naval Surface Weapons Center (NSWC)

- The plan for the acquisition and academic training of software engineers for NSWC has been approved and endorsed by their Technical Director.
- NSWC is currently implementing steps 1 and 2 (Reference appendix A).

Pacific Missile Test Center (PMTC)

- The PMTC Software Engineering Career Development Program effort is currently progressing.
- Software Management Awareness Core has been proposed and consists of the following:
    - Executive overview of Software Life Cycle Management.
    - Software Project Management Requirements and Planning.
    - Software Task Management Responsibilities for NAVAIR Projects.
- The Software Technology Update Core is in process.
    - The objective is to provide opportunity for current software employees to increase their knowledge/skills/abilities in software technology and bring productivity to optimum level.
- The Software Retraining Core Graduate Certificate Program was initiated 1 September 1980.

- The objective is to provide a core series of courses in software at the graduate level which provides expertise in software and qualify employees with a bachelor's degree to successfully complete/qualify for positions in the software function.

- The Undergraduate Software Training leading to an Undergraduate Certificate Program has not yet been approved.
- The Master of Science Program in Electrical Engineering and Computer Science has been in existence since 1975 and is designed for students who need to update their background experience and operational ability in the computer area.

Department of the Navy

- Interpretive Memorandum was issued October 1981.
- There is no indication at this time that it is in use for classification purposes.

Professional Council

The long-term solution attempted by the Council has been stymied and no new strategy has been developed. To date:

- There has been several meetings with Paul Katz, Director of the Standards Development Center.
- These meetings have resulted in no new actions to date.
- PMTC is investigating answers to the following questions and concerns raised by Mr. Katz:
    - What can PMTC do to overcome the evidence that indicates the continued requirements for "professional" qualifications?

- The use at this time of the term "engineering" in association with computer programming and systems analysis would require engineering societies and engineering granting institutions support. Such support has not been obtained. Will PMTC buttress their report to overcome these concerns?

- Are private sector employers using software engineering titles? More detailed information is needed to confirm this concern. What is PMTC's plan for contacting private sector employers to obtain this information?

- Will the establishment of a software engineering series/occupation would substantially reduce turnover among such personnel? Will PMTC conduct some kind of a pay survey and if so what is PMTC's plan for conducting such a pay survey?

Most recently, Mr. Jerry Reed, Executive Director of Acquisition for Chief NAVMAT has expressed great interest in the software problem in DOD. He is exploring what can be done at NAVMAT Headquarters to assist. His past successes allows way for a great degree of optimism.

Nevertheless, the Federal Government is at the threshold of an ever escalating software requirement. To continue to ignore the need for a positive, pro-active pursuit of a viable solution is irresponsible and can result only in disaster in the long term.

For additional information contact:

Gwendolyn E. Hunt
Associate, Systems Technology Office
Code 1201
Pacific Missile Test Center
Point Mugu, CA 93042

Appendix A

EXECUTIVE SUMMARY

The following plan was devised by the Software Engineering Committee for the acquisition and training of software engineers. This plan was presented to the Board of Directors and approved by them in October 1980.

It was recommended that all newly hired persons who are intended for organizations involved in software development, who do not already have an appropriate background, be included in the Software Engineering Development Program. There will be others who may wish to change from their present jobs to developing software who will also be included. There may be still others that management may wish to recommend for this program.

This plan assumes that the trainees have no detailed knowledge of computers but that they do have at least BA degree or equivalent in one of the scientific fields. Training opportunities will also be provided for experienced software development personnel through the series of short courses to be conducted at the Center.

Step 1. The trainees, while remaining in their present positions, will take the following three academic courses:

- FORTRAN Programming for Scientists and Engineers
- Software Development
- Computer Organization

These courses will be taught for academic credit under the sponsorship of a local university (possible sources are Mary Washington College for Dahlgren and the University of Maryland for White Oak). Appendix C contains a brief description of each of these courses.

Step 2. For those trainees who satisfactorily complete the above three courses and who desire to change work units, the Committee will assist them by putting them in touch with divisions who are in need of software engineers. At this time any further training will be planned by the trainee's supervisors. Individual Development Plans (IDP's) will be prepared for each trainee by their supervisor. This plan should include the series of short courses to be provided continuously over the next several years which will cover major aspects of software engineering. See appendix D. The IDP may also include details in other organizational units (see appendix E) which can provide experience in specific elements which support the software development process. These IDP's will be submitted to the Software Engineering Committee for approval before the trainee will be permitted to take any additional courses. The following are short courses covering the field of software engineering which will be provided by the Human Resources Development Division (P20):

- Comparative Software Engineering
- Comparative Design and Analysis Techniques
- Modern Programming Techniques
- Software Testing
- Embedded Computer System Architectural Engineering
- Software Project Management
- Software Economics
- Software Configuration Management
- Software Quality Assurance
- Software Acquisition Management

It was recommended that all of the above courses be taken by all trainees so that they will be familar with all aspects of software engineering. Other specialized courses will be added as required.

BACKGROUND

The number of tactical and strategic weapons systems projects being undertaken by the Center is constantly increasing. The amount of Center resources being applied to the development of computer software for these projects is growing at a rapid rate. In 1979 a study showed that the Center expended 610 manyears, both in-house and contractually, on tasks that resulted in operational programs delivered to the Navy and Marine Corps. A 1980 update showed that this had grown to 640 manyears. In 1978 there were about 1350 people using the Centers's mainframe computers, by 1981 this number has grown to approximately 1800.

The Bureau of Labor Statistics recently estimated that between 1978 and 1990 the number of people needed to develop computer software will more than double (Reference 1).

A National Science Foundation projection states that there will be three and one-half times as many jobs for computer professionals as there will be persons receiving the baccalaureate or master's degrees in computer sciences between 1978 and 1990 (Reference 1). Universities are just beginning to develop programs in software engineering and are not going to be able to provide the Center and other software development activities with adequate personnel to meet our needs. Appendix H contains a list of colleges and universities, known to the Committee, which offer programs in software engineering. Most of this list was taken from a study by the Professional Council of Federal Scientists and Engineers.

Center management, recognizing these trends, established a committee to advise them on the acquisition and training of software development personnel (appendix A). The "Software Engineering Committee" was formed with membership as shown in appendix B.

One of the first steps that the committee took was to make a survey of the departments to determine the perceived need for software engineers. See appendix G. This survey indicated a need for 117 additional software engineers in FY81 and a total of 373 over the next 5 years. Even though these were somewhat "off the top of the head" figures they do indicate a desperate and growing need within the departments.

SOFTWARE ENGINEERING DEFINED

Software engineering has been defined in several ways. The definition most widely accepted in the field and the one accepted by the Committee is the following:

- "The establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines." F. L. Bauer (Reference 2)

Other definitions were also considered by the Committee:

- "Science of design, development, evaluation and maintenance of computer software over its life cycle." DOD Directive 5000.29, April 26, 1976

- "The practical and methodical application of science and technology in the design, development, evaluation and maintenance of computer software over its life cycle." National Aeronautics and Space Administration

Both of the above definitions seem to say about the same thing and agree quite closely with the definition accepted by the Committee. The reason the Committee did not use one of these is that it seemed advisable to stick to one which has already been accepted by academia and most practitioners in the field.

Still another definition that is currently in use is:

- "... to ensure effective utilization of computer system resources as elements of major physical or environmental systems which incorporate one or more specific engineering disciplines. The computer systems are generally embedded within a major system complex and provide direct real-time support of and/or perform specific tasks within one or more of the system functional elements." "A Navy study initiated by CNO ltr. Ser 141c11/700569, 23 July 1980 and reported out by "Memorandum for the Record from the Office of the Chief of Naval Operations, Navy Study Group, 'Definition of Software engineer,' August 29,1980."

This definition, derived by the Navy Study Group, was proposed in an effort to give guidelines for Navy position classifiers in classifying software engineering-type positions in other professional series. The problem that this Committee had with the above definition is that it does not recognize the fact that all large software systems must be "engineered," i.e., "sound engineering principles" applied in its development, whether for embedded computer systems or for software systems intended for general purpose computer. The same principles of software development apply.

A federal task force on the west coast defined software engineering as:

- "...professional engineering work in design, development, test, evaluation and maintenance of software, and software and hardware as an integrated entity, for computer systems which control equipment and/or respond to physical stimuli." (Professional Council of Federal Scientists and Engineers, OPM San Francisco Region , October 10, 1980)

The group which used this definition was developing a proposed standard for the classification of civil service positions in the government. The trouble the Committee had with this definition was two-fold. First it is restricted to embedded computers, and secondly, it would require an engineering degree. Members of the Committee met with the chairperson of this group in an effort to work out an acceptable compromise because this definition will eventually find its way into the new classification standard series description of software engineering. As a result of these meetings, the Professional Council of Federal Scientists and Engineers removed the restriction to embedded computers, but retained their proposed classification in the 800 series, thereby still requiring an engineering degree.

The field of software engineering covers all phases of the software development process.

DISCUSSION

The Committee realized immediately that no amount of formal training that we could define could produce expert software engineers, but that the most we could hope to do was to give people enough knowledge to get them started in the field. On-the-job experience could then possibly make them "experts."

The Committee reviewed and accepted the 13 knowledge areas identified as being critical to software engineering in a previous study regarding the academic preparation of software engineers (reference 4). These knowledge areas are defined in appendix F and are as follows:

- Controls
- Process exposure
- Design principles
- Communication skills
- Functional capabilities of digital hardware
- Software design technology
- Evaluation
- Systems integration
- Programming techniques
- Human factors
- Information structures
- Communications technology
- Systems simulation

This study indicated that many of these areas are best learned through on-the-job training. The Committee selected three core courses as providing the necessary foundation for learning software engineering. See appendix D. It was decided that these courses should be taught under the auspices of a college so that the trainees would realize that they will have to put some effort into the training since homework will be assigned and grades will be given. It was also hoped that getting college credit for the courses would encourage some of the trainees to persue a degree in a computer related field.

It was decided that the trainees who might wish to change career fields and go into software engineering should stay in their present jobs while taking the core courses. This was so that if anyone did not succeed in the courses that they would still be in an appropriate job. It was also felt that any amount of knowledge they did gain from the courses would be beneficial in any job in an R&D organization such as NSWC.

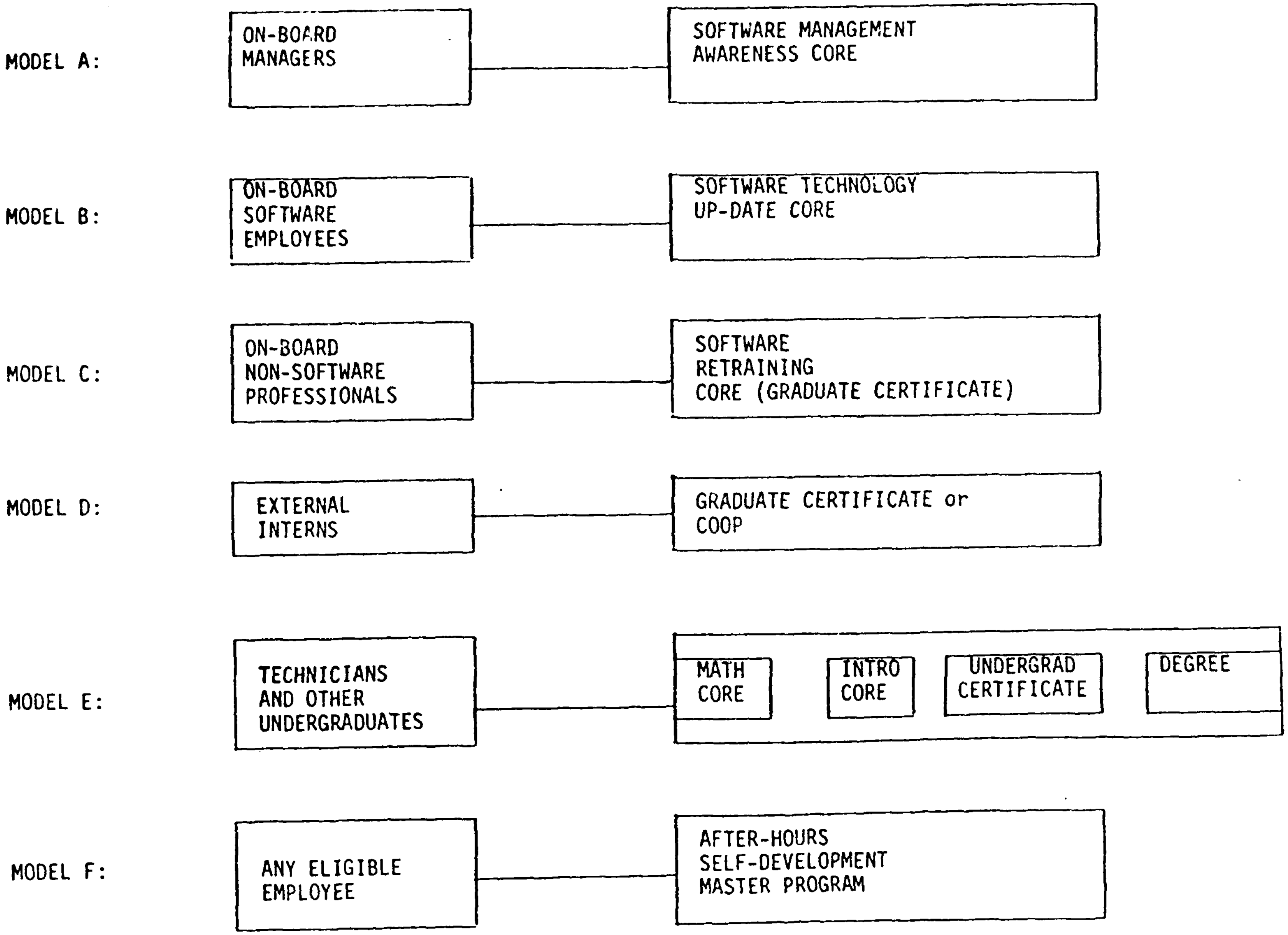
It was recommended that this training plan be restricted to holders of scientific degrees in order to have a somewhat homogeneous background and scientific maturity in the classes. It was recognized that there will be others without degrees who may have the capability of succeeding in the program and they will be considered on an individual basis.

The above recommendations and decisions were approved and supported by the Board of Directors and the Technical Director.

An area where the Committee feels more effort is required is in the [illegible] of the differences between software engineering and systems engineerin similar to the distinction that now exists between electronic engineeringg and systems engineering.

Appendix B

PMTC SOFTWARE ENGINEERING DEVELOPMENT PROGRAM


MODEL A:

ON-BOARD MANAGERS — SOFTWARE MANAGEMENT AWARENESS CORE

MODEL B:

ON-BOARD SOFTWARE EMPLOYEES — SOFTWARE TECHNOLOGY UP-DATE CORE

MODEL C:

ON-BOARD NON-SOFTWARE PROFESSIONALS — SOFTWARE RETRAINING CORE (GRADUATE CERTIFICATE)

MODEL D:

EXTERNAL INTERNS — GRADUATE CERTIFICATE or COOP

MODEL E:

TECHNICIANS AND OTHER UNDERGRADUATES — MATH CORE | INTRO CORE | UNDERGRAD CERTIFICATE | DEGREE

MODEL F:

ANY ELIGIBLE EMPLOYEE — AFTER-HOURS SELF-DEVELOPMENT MASTER PROGRAM

Figure 1: PMTC's Complete Modular Software Engineer .raining Program

MODULE A

SOFTWARE MANAGEMENT
AWARENESS CORE
(PROPOSED)

This module provides for orientation in software technology for new managers; and update training for supervisors/managers who have worked in the functional areas or pheriphery of software. The module is designed to apprise these managers of current problems; new procedures, regulations in the acquisition process; recent technology; and future trends. The course work includes.

EXECUTIVE OVERVIEW OF SOFTWARE LIFE CYCLE MANAGEMENT

OBJECTIVE: To provide executives a management overview of software with emphasis on integration of NAVAIR policy and its impact/implementation at PMTC.

SOFTWARE PROJECT MANAGEMENT REQUIREMENTS AND PLANNING

OBJECTIVE: To provide an expanded overview of course 1, with emphasis on definition, review and acceptance of software deliverables, integration within project functions, and software support interactivity relationships, as well as the overall project management functions including contract/procurement and budget.

SOFTWARE TASK MANAGEMENT RESPONSIBILITIES FOR NAVAIR PROJECTS

OBJECTIVE: To provide similar overview as outlined in courses 1 and 2, with specific "how-to" information, and discussion of interrelationships between PMTC employees and NAVAIR or CONTRACTORS.

MODULE B

SOFTWARE TECHNOLOGY UP-DATE CORE
(IN PROCESS)

OBJECTIVE: To provide opportunities for current software employees to increase their knowledge/skills/abilities in software technology and bring productivity to optimum level.

A task/competency survey in the occupations requiring software expertise conducted in late FY-77 resulted in the identification of 25 courses. These courses ranged from basic introductory/orientation training for new employees; intermediate theory/concept/application courses for employees needing refresher or exposure to new software trends/applications; to highly specialized technical courses relating to tactical fighter weapons systems. These courses are presented on a cyclical basis, and revised as technology changes.

Automatic Data Processing Orientation
Introduction to Computer Programming
Introduction to COBOL Programming
Introduction to FORTRAN Computer Language
Computer Communications Systems
Computer Communications Networks
Data Communication Systems and Networks
Software Design for Data Communication Systems
Modern Digital Communications
Configuration Management of Software Programs
Communications Systems Engineering
Structured Programming Workshop
Spread Spectrum Communication Systems
Design and Applications of Computer Graphics Systems
Small Computer Systems
Structured Design and Programming
ECM and ECCM for Digital Communications
A Modern Approach to Control of Processes, Vehicles and Large Scale Systems
Computer Performance Evaluation
Microprocessor Approach to System Design
Digital Continuous-System Simulation
Command and Control for Air Force, Army and Naval Systems
Computer-Aided Decision Making
Avionics Systems Engineering
Computer Security

MODULE C
SOFTWARE RETRAINING CORE

GRADUATE CERTIFICATE PROGRAM
(STARTS 1 SEPT 1980)

OBJECTIVE: To provide a core series of courses in software at the graduate level which provides expertise in software and qualify employees with a bachelor's degree to successfully compete/qualify for positions in the software function.

This module was planned to provide opportunitues for professional employees holding a degree in engineering, math, physics, or computer science, and allows participation in a graduate certificate program which would qualify them to be reassigned into positions requiring software engineering experience.

The Graduate Certificate Program is a core of engineering and computer courses which allow professional employees having, at a minimum, a bachelor's degree in one occupation an opportunity to expand their knowledge/skills in the software area. The core consists of eight courses; however, dependent upon the individual's background and the courses previously taken, the training is determined on an individual basis. For example, if an employee had taken two of the core requirements previously, and can substantiate grades, it would not be necessary to repeat the courses. Further, if the individual has not had pre-requisite courses, it is a requirement that all pre-requisites are met prior to enrollment in the Graduate Certificate Program.

When requests for retraining are received in the Civilian Personnel Office, a counseling session is established with specialists in the software function and a representative from the Civilian Personnel Office, who determines the current level of knowledge of the individual and program the additional certificate courses required to qualify as a software engineer.

This module can also be utilized during Reduction-in-Force periods for retraining employees whose positions have been abolished and where resource requirements for software employees are still needed, giving management greater resource flexibility. With minimum training time, resources become available from within an organization when difficult recruitment or retention problems create an environment less than adequate to satisfy workload requirements.

The program is designed to serve personnel working in the areas of engineering and computers who already have bachelors or higher degrees, but who, because of the rapid development of technology, wish to advance their knowledge or to develop competence in computer ergineering without undertaking a formal degree program. Required for completion of the program are five courses, of which a minimum of three must be graduate courses.

Courses included in the program at Point Mugu:

ECE 152A: Logic Design & Switching Theory

ECE 152B: Digital System Design I

ECE 152C: Digital System Design II

ECE 154: Introduction to Computer Architecture

COMP SCI 130A; Data Structure and Algorithms

COMP SCI 162: Principles of Programming Languages

COMP SCI 170: Introduction to Operating Systems

COMP SCI 174: Data Base Management System

ECE 250: Advanced Topics in Computer Architecture

ECE 252A: Sequential Machines and Automata Theory

ECE 252B: Digital Computer Design and Arithmetic

ECE 252C: Advanced Topics in Digital System Design

ECE 256: Digital Design Automation

ECE 257: Fault Tolerant Computing

COMP SCI 272-B: Software Engineering

COMP SCI 274: Advanced Topics in Data Base Management Systems

ECE 279: Computer System Performance Evaluation

MODULE D

GRADUATE INTERN PROGRAM

To be arranged at a later date

MODULE E

UNDERGRADUATE SOFTWARE TRAINING
LEADING TO:
UNDERGRADUATE CERTIFICATE PROGRAM
(NOT YET APPROVED)

This module uses the concept of Upward Mobility and self-development. Training is received through progressive phases of the training module, culminating in a software engineering degree. This module is an attempt to "grow our own," and is based on the assumption that employees in low-paying or dead-end positions are willing and eager to acquire skills/knowledges in a career field having potentially a higher career progression pattern than the occupations in which the employees are currently performing.

The first two submodules are designed to provide, initially, expertise in the math and digital logic skills and the determination of aptitude for the career field. These submodules are offered with heavy math and introduction/orientation of skills/knowledges required to perform as a software engineer. It is anticipated that there will be a heavy mortality rate in these two submodules; however, if even 50% complete the two modules successfully and are still interested in pursuing a degree, it will provide additional input and resources for the functional areas needing aide or technician skills.

The third submodule represents the core requirements which enable the student to obtain an undergraduate certificate in software science. This module is designed for the student to receive progressive education/expertise in the broad bands of digital engineering knowledges required to function in the basic software area. If the student completes the modules through fruition of the certificate program, he/she could request assignment into one of the software functional areas at PMTC. The remainder of training required to obtain a bachelor's degree in software engineering would become the fourth submodule.

After successful completion of the modules through the certificate program if the student is desirous of continuing the educational process and receive a degree, specific counseling and individualized program design would be coordinated with the educational institution sponsoring the training program. Student short-term and long-term goals would be reflected in an Individual Trainee Development Plan, approved by the supervisor. The IDP would reflect on-the-job assignments in a functional software area, and would essentially create a work-study environment, making practical application of learned concepts more meaningful to the student. Management would acquire additional day-to-day resources which could begin as an aide and grow through technician skills as the student becomes more proficient. This concept allows for additional resources in a skill-shortage high-demand area over a longer period of time, with the on-the-job assignments designed specifically to enhance the student's knowledge/skills and assist in meeting functional requirements. Academic course costs would be included in the cost center's annual budgets.

POSSIBLE CURRICULUM MATRIX

GROUP I	GROUP II	GROUP III	GROUP IV	GROUP V
Albebra	Intro to Computers	FORTRAN	Bachelor's degree courses (to meet specifications of Lead Institution).**	Postgraduate courses (UCSB or other)
Geometry	Digital Circuits	COBOL		
Trigonometry	Boolean Algebra	Data Base Management Systems		
Statistics	Digital Logic	Query Languages		
Accounting	BASIC	Display Mgmt Systems		
Calculus	Introductory Assembly Language	Communications Software		
Intro to Programming*	Digital Communication			

* An early introduction to programming may well prove useful. Programmable calculators could be used as early as the introductory algebra course.

** A thorough survey of Lead Institution and postgraduate requirements is needed before specific modules are designed, in all groups.

MODULE F

MASTER OF SCIENCE PROGRAM
IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

This module has been in existence since 1975, and is designed for students who need to update their background, experience, and operational ability in the computer area. It is a part-time degree program for students who, because of employment responsibilities, cannot attend the main campus or pursue full-time study. It provides opportunities for students to broaden their understanding of both the design and operational characteristics of modern computers and computer systems. The program offers a variety of topics from digital system design to software architecture, enabling students to pursue study ranging from the broad implications of new developments in the field through specific aspects of hardware or software theory.

The delivery system for this program is somewhat unique. Training is presented via a microwave TV link, with one-way visual and two-way audio, which allows for real-time interaction between student and professor. At present, the program, consisting of 42 quarter units, is designed for students who need to update their background, experience, and operational ability in the computer area. It provides opportunities for students to broaden their understanding of both the design and operational characteristics of modern computers and computer systems. Since student needs are varied, the program explores a variety of topics from digital system design to software architecture, enabling students to pursue study ranging from the broad implications of new developments in the field through specific aspects of hardware or software theory.

Courses included in the Program:

*Comp Sci 120A:	Automata & Formal Languages
ECE 140:	Probabilistic Methods of Signal and System Analysis
*ECE 152A:	Logic Design & Switching Theory
ECE 152B:	Digital System Design I
ECE 152C:	Digital System Design II
*ECE 154:	Introduction to Computer Architecture
*Comp Sci 130A-B:	Data Structure and Algorithms
ECE 250:	Advanced Topics in Computer Architecture
ECE 252A:	Sequential Machines and Automata Theory
ECE 252B:	Digital Computer Design and Arithmetic
ECE 252C:	Advanced Topics in Digital System Design
ECE 256:	Digital Design Automation
ECE 257:	Fault Tolerant Computing
Comp Sci 260:	Advanced Topics in Translation

*A total of eight units selected from these four starred courses may be used for credit toward the master's degree.

Comp Sci 160A: Translation of Programming Languages

Comp Sci 162: Principles of Programming Languages

Comp Sci 170: Introduction to Operating Systems

Comp Sci 174: Data Base Management System

Comp Sci 262: Languages

Comp Sci 270A: Advanced Topics in Operating Systems

Comp Sci 272A-B: Software Engineering

Comp Sci 274: Advanced Topics in Data Base Management Systems

ECE 279: Computer System Performance Evaluation

Appendix C

COUNCIL EFFORT

THE PROFESSIONAL COUNCIL
OF FEDERAL SCIENTISTS AND ENGINEERS
c/o OFFICE OF PERSONNEL MANAGEMENT
525 MARKET STREET
SAN FRANCISCO, CALIFORNIA 94105

In Reply Please Refer To
1201/12000

ADVISORS TO THE DIRECTOR, SAN FRANCISCO REGION, OFFICE OF PERSONNEL MANAGEMENT

From: K. I. Lichti, Chairman, Software Engineer Standards Study
To:

Subj: Software Engineering Study

Encl: (1) Questionnaire (Software Engineering Preliminary Study)

1. I am writing to ask for your activity's participation in a preliminary occupational study of Software Engineering being conducted by the West Coast Region Professional Council of Federal Scientists and Engineers. FPM Bulletin 271-35 identified that the Standards Development Center, Office of Personnel Management (OPM), Washington, D. C., planned to undertake such a study during FY 80 when resources became available. For several years the Council has had a strong interest in the development of classification standards for this emerging occupation. With technical assistance from OPM, the Council has undertaken this preliminary study to develop input to the Standards Development Center project.

2. Enclosure (1) is a brief questionnaire which I am asking you to complete and return by August 1, 1980. A coordinated response between your technical staff and the personnel office servicing your activity is encouraged. Your reply should be sent to:

Ms. Gwendolyn E. Hunt
Associate Head, Systems Technology Department
Code 1201, Building 512
Pacific Missile Test Center
Point Mugu, California 93042

All responses will be used for this study purpose only. Please call Ms. Hunt at commercial 805-982-7552, or AUTOVON 351-7552, or Ms. Julie Bennison of the Classification and Compensation Branch, Western Region, OPM, at commercial 415-556-4759, or FTS 556-4759 for further information.

3. Your participation is critical to obtaining the data necessary for a successful study. We thank you in advance and will provide you with summary information of our study results.

Sincerely,

K. I. LICHTI

QUESTIONNAIRE
(Software Engineering Preliminary Study)

Series Description: Software Engineer GS-8XX

This series includes positions the duties of which involve professional engineering work in design, development, test, evaluation and maintenance of software, and software and hardware as an integrated entity, for computer systems which control equipment and/or respond to physical stimuli. Performance of the work requires application of knowledges of: (a) professional engineering concepts, principles, techniques, and practices; (b) computer system architecture, languages and software systems; and (c) mathematical and statistical techniques.

1. How many positions at your activity would be covered by this tentative software engineering series description? How are these positions currently classified? Please provide position descriptions and functional statements for the organizations where the positions are located.

2. What do you consider the minimum qualification requirements for placement into an entry-level software engineering position? If you identify an educational level as the minimum qualification, are there alternates of experience or experience plus education which you would consider comparable to the educational level? If no, please explain why not.

3. What criteria do you apply for selecting employees to enter into or to advance in this occupation?

4. What are the principle knowledges and skills used in full-performance level assignments?

ENCLOSURE 1

5. What distinguishes software engineering from:

Electronics Engineering GS-855?

Computer Specialist GS-334?

Computer Scientist GS-1550?

Mathematician GS-1520?

6. Are you aware of education or formal training programs for software engineering? If so, where?

7. Do you know of comparable jobs in industry? If so, what companies and locations?

8. Additional comments?

RESPONSES

1. Statistics by Department are included as a matter of interest, and since it was readily available to us. We attach no particular significance to presentation by Department since our interest is directed toward the total rather than the individual parts.

2. The 18% questionnaire response is not considered indicative of low interest. Some installations consolidated their results into one questionnaire. A better indicator of interest is the 26% installation response. Regardless of the level of interest indicators, responses indicate that approximately 1200 positions are directly affected by any conclusions drawn from this study.

3. Response by the Navy was significantly greater than all others. NASA results are skewed due to relatively few questionnaires sent and responses received. Percent of response does not necessarily increase in relation to the number of positions affected.

4. Ninety-seven percent of all affected positions are currently classified to a professional series. Eighty percent of the identifiable positions are classified to the GS-855 series. (See enclosure (2).)

	Navy	Air Force	Army	NASA	Other	Total
Number of installations contacted	25	26	5	2	11	69
Number of installations responding	10	6	1	1	1	19
Percent of installations responding	40%	23%	20%	50%	9%	28%
Questionnaires sent	106	117	33	12	17	285
Questionnaire responses	34	8	5	3	1	51
Percent of response by questionnaire	32%	7%	15%	25%	6%	18%
Positions affected	*932	276	23	8	X	1239

*Does not include approximately 100 GS-334 positions at the Naval Regional Data Automation Center, Washington, DC, which were identified as traditional GS-334 positions rather than software engineers.

CURRENT CLASSIFICATION OF SOFTWARE ENGINEERING POSITIONS

Series	Number of Positions
GS-334	43
GS-801	4
GS-830	4
GS-855	417
GS-861	2
GS-1301	1
GS-1310	11
GS-1520	36
GS-1550	5
TOTAL	523
Not identified by series but as Professional, primarily Engineering	716
GRAND TOTAL	1239

ENCLOSURE 2

MINIMUM QUALIFICATIONS REQUIREMENTS FOR SOFTWARE ENGINEERS

Responses to question 2, which asked for identification of the minimum qualifications requirements for placement into entry level software engineering positions, covered both educational and experience requirements. The analysis team tentatively concluded, based upon both general and specific responses, that the proposed occupation requires a professional degree for entry level. Of approximately 50 responses, 35 or 70% indicated any professional degree as a minimum requirement. The following chart indicates the results of all responses:

Minimum Qualifications

Agency	Engineering Degree Only	Any Professional Degree	Any Degree	All Others
Navy	9	23	1	5*
Air Force	4	4	0	0
Army	0	5	0	2*
NASA	0	3	0	2*
TOTALS	13	35	1	9*

A large majority of responses indicated that the occupation would be professional, and that an engineering degree probably would be preferred, however, not necessarily required. The analysis team concluded that the entries reflecting training (less than a BS degree) plus 2 to 3 years of experience, probably were more applicable to technicians than professionals or reflected training which could lead to, or be part of a continuing education program designed to provide full performance skills.

*Indicates that training plus experience is an alternative method for entry into occupation.

This series includes positions the duties of which involve professional engineering work in design, development, test, evaluation and maintenance of software, and software and hardware as an integrated entity. Performance of the work requires application of knowledges of: (a) fundamentals and principles of professional engineering; (b) computer hardware, software and systems operations; (c) mathematics, statistics and applied science; and (d) resource and project management.

This is a new series coverage standard. It provides occupational information to clarify the intent of the series definition. Many positions that fall within the series were formerly classified to the Electronics Engineering Series GS-855. Other occupations also closely identified with this series include Computer Science Series GS-1550, Mathematics Series GS-1520 and Physics Series GS-1310. Inclusion in this new series of positions now coded to the latter three series will depend upon the extent to which professional engineering knowledges are also required.

SERIES COVERAGE

Software engineering is an emerging occupation within the professional engineering disciplines. The hallmark of the work is the application of the scientific discipline and professional engineering principles to software system design, development and management. The work is performed to ensure the effective utilization of the full range of computer resources as elements of operational systems. The operational systems or project may be a new ship, an on-board command and control system, an aircraft or missile system, an automated air traffic control system, a range sensing and tracking system, or a subsystem or component element within a subsystem such as the guidance or firing system of a missile. The computer resources are the computer equipment (hardware), computer software (application and operating system), and all other related elements such as documentation, contractual services, personnel, supplies and facilities.

Software engineering positions are found principally in military agencies which require unique equipment and systems to be developed and maintained over the total life cycle, either by the agency or by contractors under the agency's oversight. Other agencies are also developing requirements for engineering of computer systems in equipment and facilities such as that used at airports, for modeling and simulation, etc. Software engineers perform a variety of functions dependent upon the mission of the activity and the phase of the project that they are engaged in. The missions of the activities may be research and development, acquisition management, and/or operational support to installed systems. Typical functions performed may include:

--analyze total system requirements of a major operational system to establish software functional reqirements and identify tradeoffs between use of computer resources and other special purpose hardware;

--perform conceptual design and planning by analyzing system software requirements as elements of total system design;

--formulate and conduct demonstrations to validate design concepts and to provide a basis for acceptance of the software system for full-scale engineering development and subsequent production;

--conduct development of design criteria and actual design of software systems, evaluating and monitoring contract work, and establishing test requirements and validation procedures which measure software reliability and ensure that initial requirements have been met;

--ensure that state-of-the-art software engineering methodology is used to implement the software system design; and

--participate in the revision, update and enhancement of installed software/firmware.

The systems approach prevalent in the Federal Government today recognizes that software is frequently the major area of system cost and the focal point of overall system effectiveness. The engineering principles which have traditionally been applied in hardware development to analyze and prove reliability and efficiency are being applied equally to software development, and are replacing the empirical methods used in the earlier phases of the software industry. The undisciplined approach to development of the elements of a computer system is more and more being replaced by the systems approach, so that hardware and software and interfaces thereof are designed, developed, tested and evaluated as an integrated project. Computer technology is also changing. The distinction between software and hardware is merging through increasing applications of firmware. In the context of these changes, the software engineering occupation has formed.

The engineering methodology provides a disciplined, systematic approach to problem solving and systems design as well as the basic knowledge essential to understanding the physical aspects of the system hardware and its inherent capabilities and limitations. Project engineering including requirements analysis, cost leadtime estimating, life-cycle support planning and economic analysis of alternatives are engineering functions required for software systems development. Development and validation of mathematical models require a thorough understanding of mathematical techniques including discrete mathematics, theory of optimization, simulation methods, etc. Computer science is required to insure that computer system architecture, language theory, fundamentals of operating systems, compilers and assemblers and programming practices have been applied to maximize the systems efficiency and reliability, and integration of hardware and software functions. Management skills are essential for planning and conducting development efforts requiring coordination of diverse support efforts contributed by others. Thus, the software engineer brings both depth and breadth of knowledges and experiences from several disciplines--engineering, mathematics and computer science--to ensure tht software systems utimately represent the most effective implementation from the total systems point of view.

EXCLUSIONS

The following types of positions are excluded from this series:

1. Professional positions that involve research or development in computer science. Such positions are classified in the Computer Science Series GS-1550.

2. Positions that require the application of professional engineering knowledges and principles in reference to electronic circuits, equipment and systems. Such work does not require the significant computer technology knowledges of software engineering. These positions are classified in the Electronics Engineering Series GS-855.

3. Positions which require full professional qualifications in mathematics. Such work does not require application of engineering fundamentals or digital computer technology. Such positions are classified in the Mathematics Series GS-1520.

4. Positions which involve design or implementation of systems for solving problems or accomplishing work processes by the use of digital computers. Such work does not require application of professional engineering of scientific knowledges and is typically oriented to transactions in business, management and industrial functions. Such positions are classified in the Computer Specialist Series GS-334.

TITLES

The title Software Engineer is authorized for nonsupervisory positions at all levels in this series. Positions which involve supervisory duties and responsibilities as defined in the Supervisory Grade-Evaluation Guide are identified by the prefix "Supervisory."

GRADE LEVEL CRITERIA

The series coverage standard does not provide grade level criteria. Selection of the standard(s) to be applied should be based on the principle duties, location and function, and knowledge requirement of the position to be evaluated. Some of the standards which may be useful include the Equipment Development Grade-Evaluation Guide or the Test and Evaluation Engineering Grade-Level Guide; or Electronics Engineering GS-855. Supervisory positions should be evaluated by reference to the Supervisory Grade-Evaluation Guide.

DISTINGUISHING CHARACTERISTICS OF SOFTWARE ENGINEERING

From GS-855 Electronics Engineering:

- lacks computer hardware and/or software knowledges (21 respondents)

- is oriented to systems, i.e., non-computer, hardware (16)

- the Electronics Engineer is to hardware as the Software Engineer is to Software (4)

- Software engineering is the same as electronic engineering (5)

From GS-1520 Mathematics:

- lacks engineering fundamentals and principles (19)

- lacks computer hardware and/or software knowledges (17)

- is oriented to a theoretical rather than an applied approach (7)

From GS-1550 Computer Science:

- lacks engineering fundamentals and principles (11)

- limited to research and development functions, and hence, is oriented to theoretical rather than an applied approach (10)

- software engineering is the same, that is, is a subset of the computer scientist occupation (4)

The above is a compilation of the major responses on the distinguishing characteristics of software engineering. Though there were other varying responses not listed above, we found them to be singular and not susceptable of grouping as a significant comment. Clearly the lack of engineering fundamentals is the primary deficiency in the non-engineering series, and the lack of computer knowledges is the primary deficiency in the engineering series. Correlating this data to that given on minimum qualifications, it is noted that many of the respondents who stated that a degree in electrical engineering was the necessary academic preparation coupled that with course work in computers. It is also significant to note that some responses matched software engineering totally with GS-855 and others matched totally with GS-1550. The majority response, viewed as a whole, did not find any of the single currently established series to fully encompass all of the required knowledges of software engineering.

CONCLUSIONS

There is strong agreement that the nature of the work is professional.

It is not established that an engineering degree is an absolute requirement, although the responses indicate that training in engineering fundamentals and principles is an essential aspect of the academic training. Where a response indicated that minimum qualification requirement is for an engineering degree or a computer science degree, it was recorded as reflecting "any professional degree" not exclusively engineering. However, to the extent that computer science degrees are in fact offered as part of the engineering department, this response may more accurately reflect engineering only as the essential requirement.

The knowledges, skills, and abilities identified show a consistant pattern even though there appears to be some flexibility in the particular academic discipline which provides these knowledges, skills, and abilities.

There are minority opinions. The questionnaire responses from the Naval Weapons Center, China Lake, and the Naval Surface Weapons Center, Dahlgren, illustrate some of the diverging viewpoints on the scope of the definition and basic qualification requirements. There are also differing viewpoints on titling reflected in these and other questionnaire responses, with suggestions such as "Computer Engineer."

BIBLIOGRAPHY

1. Paul L. Anderson, Capt. USN, Commander, "Report of the Software Engineering Committee," Naval Surface Weapons Center, Dahlgren, Virginia © August 1981.

2. Randall W. Jensen, Charles C. Tonies, Software Engineering, Prentice-Hall, Inc., Englewood Cliffs, New Jersey © 1979.

3. Thomas R. Muir, Special Assistant for Civilian Personnel/EEO, "Department of the Navy Interpretive Memorandum," Navy Publications and Forms Center, Philadelphia, Pennsylvania.

4. Naval Material Command: Master Plan for Tactical Embedded Computer Resources, Washington, D.C., 31 July 1978.

5. United States Civil Service Commissions: Position Classification Standards, March 1957.

A NEAR REAL-TIME SYSTEM FOR THE REDUCTION/PRODUCTION OF CINETHEODOLITE DATA

Presentor

James R.S. Manion
Naval Weapons Center
China Lake, CA

ABSTRACT

Although the typical objections to film data center around the labor intensive, time-consuming aspects of film processing and reading, our analysis revealed that the chief impediment to the timely and economic delivery of a data product was our batch-oriented, large-mainframe mode of operation after the film was read. This paper documents the approach and implementation of an integrated hardware/software system dedicated to the collection, reduction and production of data within tens of minutes after the completion of the reading task.

Perhaps it's only fair to begin this talk with a brief description of what cinetheodolites are. Cine refers to motion pictures and a theodolite is an angle measuring device, so a cinetheodolite is a motion picture camera which also records azimuth and elevation information, i.e., the direction which the camera was pointing, on the film frame. So if one captures an object on two cine-T's simultaneously, and knows exactly where the cameras are, he can deduce the object's location. In fact, two frames and you have acceleration. Of course, there are various ways of mathematically handling the rays--they never quite intersect at a point--but for the moment we'll assume they do.

Now there's a huge problem--with everyone who has cinetheodolites--involving the production of data. It tends to take a long time. One way to speed things up is to omit the reading--that is to avoid locating the object of interest with respect to the center of the film frame. Usually there's about half a degree field of view on the frame, so if you accept just the angle readings and don't consider the tracking, your accuracy from each instrument goes from ±.005 degree to ±.5 degree or at say 10,000 feet slant ranges from approximately 1 foot to 100 feet.

Raw cine-T data consists of records with five or six numbers per film frame, namely, time, frame number, X- and Y-counts of the object of interest to the center of the film frame, azimuth dial reading and elevation dial reading. Under the old system these records went on the CCF (Central Computing Facility) mainframe via tape and began a 2- to 3-week data cycle, which was both expensive and frustrating.

Two years ago we learned that the CCF was going to replace the main computer, perhaps of a different manufacture, by summer 1982. This meant that we would either have to do an extensive rewrite of the existing poorly documented software, which was written in the 1950's, or seize the opportunity to clean up and speed up the entire cycle by putting on a minicomputer. So by spring of 1980, which is when we really were getting serious about this, 90% of the hardware was procured and we started to forge ahead.

The software development project I'm about to describe to you was completed in 18 months and right on schedule, a bit under cost. We're pretty proud of it, so elements of bias may surface, but I'll try to stay factual here. I'd like to be able to relate to you six or eight really neat principles that will always work in software development. If they exist we surely haven't discovered them.

What we did learn was that data basing as a method of documentation and data structuring paid off. We talked with film readers, range customers, etc., a lot for the first 6 months and didn't write any code, except for the data base schema. After 6 months and nothing concrete done, we got scared and wrote a lot of code in the last 12 months,

So in late 1979 we addressed ourselves to the problem of creating a high quality data collection, reduction and production system in near real time. We felt that instead of making the customer wait for from 1 to 3 weeks from the time the film is read we could give him a quality data product within 10 minutes. And we certainly felt we could reduce costs. We were then transferring a quarter of a million dollars to the mainframe every year in file charges, run time charges, page charges, that kind of thing. That's a considerable amount of money--the rates are quite high.

All of this hardware I am going to describe to you cost a little over $100K, so you can see we are amortizing our investment, even including the cost of software, quite rapidly. The cost of operating the system--hardware and software maintenance, consumables, utilities, labor--should be somewhere in the neighborhood of $30K.

The current system is also very labor intensive. We're transferring about a quarter of a million dollars to the contractor just to read the film and force data through this inefficient data cycle. We expect to see great savings here and we hope, and certainly expect, that the software maintenance will be much less burdensome, since the code was written along much more structured and better documented principles.

In 1979, we were looking at the mainframe being decomissioned in 36 months. Now we're looking at the mainframe being decomissioned in 2 months, so they're right on schedule--in fact a little ahead of schedule and so are we.

So, to summarize the old software--most of it 20-year old FORTRAN II without a comment in it--is unverifiable, unmaintainable and inefficient in both labor and machine resources. We also had security problems which we no longer have running a self-contained, securable mini. Everybody is talking these days about crypto and transmitting secure data, but these considerations are largely eliminated if you get a dedicated computer at the site where the classified data is produced.

One of the main problems with film reading is quality control, and I don't use the new term quality assurance--I use quality control, The best we can hope to do is control mistakes--we sure can't assure that there are none. So, we decided early on that there is one way that we could make a major improvement in the current system, that as the film is being read, we could be predicting ahead, just as on the basis of the angle data, we could be predicting ahead what the next reading should look like, and defining a window about that predicted value. If the next reading falls within that window, we light the little green light. If it falls outside of that window, we light the little red light and ask for that frame to be re-read--just once. We feel that we are catching about 80 to 90 percent of the film reading errors with the real-time quality control, and that's a rather significant improvement there.

So we set, as a performance criterion, that we would have real-time quality control for the film reader. As the person is reading the film and entering the azimuth, elevation, time, positioning the cross hairs, that within 1 second, we would tell that person if the reading fell within the prediction window or not. We have met that with our system.

We decided that the system should be data driven, and by that I mean, when the film is edited, looked at before it is read. The editor enters into the system what flight-pass-object he wants to call these things, and which cameras are viewing the event, and so on. He enters all this kind of data in an interactive, prompted fashion. Now the system knows--and I'll show you what I mean by knows--but the system knows now what to expect. You then give the film to the film readers, they read it, and as soon as the last frame of film is read and verified as being correctly read, the data drives itself through the system. The camera angles, without any further intervention, produce trajectories. The trajectories, without any further intervention, produce the graphs that the editor said would be required when he edited the film.

We feel that this cuts down on labor, cuts down on the amount of human interventions. and consequently the number of error sources. Of course, it produces funny results sometimes too because it kind of starts a little prematurely. With that in mind, we were looking and expecting a 10-minute data product and a 22-millisecond solution. I will talk about that a little later. One of the big questions was, "Is that n-station solution--that program that takes the angle data and gives you solution point-data--is that thing going to down-load the resources? Can a minicomputer really do that?" There were a lot of people who told me that was a really crazy idea, and that you had to do that on a mainframe.

I decided I better quick code up that solution and run it, to see if I should back out gracefully, or carry on. When I did that, and ran through, Hewlett-Packard has a nice package for determining the run time of a particular chunk of code called the Profile Monitor. I found that that solution runs in 22 milliseconds, so that obviously was not a concern. What consumes the resources on the system is the data base management system, to the extent that anything consumes the resources. But we wanted to make darn sure early on that that solution could run in a timely fashion.

A real-time quality control aspect works something like this. When the film reader sits down to read the film, he reads the first two points of the first two frames of film, and they are accepted uncritically. This primes the predictor. At that point, the filter/predictor then predicts what the third frame should look like in azimuth, elevation and time, and it also sets a window, which is just a nominal window at this point, of ± so many thousandts of a degree, within which it will accept readings. And, when the third frame is read, if it falls within that window, it lights the little green light. We did have to develop a box, which is about as big as a chalk eraser, which goes right above the keyboard on the film reading machine, and there is an azimuth, elevation and time light on it. There are red lights, and if the machine doesn't like the azimuth, it lights the azimuth light; if it doesn't like the elevation it lights the elevation light; and so on. If it likes all three readings, then it lights the little green light. It also has a little buzzer which beeps one pleasant little beep if it liked the reading, and two rather squawkish beeps if it didn't. If you can get away with not developing hardware, do it. That's the simplest piece of hardware I have ever seen. But we did it ourselves, and that consumes a lot of resources. However, if you own a minicomputer, it is hard to really stay away from it. We had to solder our own cables and really learn more than I wanted to know about communications protocols and stuff like that, so you do get stuck with it. To be effective with a minicomputer, you do have to learn a little about input/output (I/O), but it certainly is a great way to consume resources and get off schedule, and that kind of thing. So that's what lights the lights and tells the reader whether you want to re-read the current frame or go on to the next frame.

The orients have always been a big problem with us. They are Q.C'd as well. In order to interpret data, you have to shoot orientation lights. So before or after you shoot the real data, you shoot a few fixed points, tops of mountains, stuff like that, lights. Typically, the film reader will tell the system that that orientation light was one light, when in fact it was a different light. Obviously then, it would do the wrong things with that data and give you the wrong answers. So, what we have done is--we have quality controlled the orientation. We store in the system the approximate azimuth of a given orientation light when viewed from a particular camera. When the person then reads that orientation light, it isn't accepted, but it's checked against that morgue value of what should be seen from that camera, and it is again Q.C.'d and the lights are lit if the system doesn't agree with those readings. That saves us a heck of a lot of re-running.

The interactive EDLOG program is the one where the editor sits down and enters all the data about the test into the system before the film is read. This is a nice friendly program--you can't hurt it. You can always backup, you always have a doublecheck after you enter the values, it throws them all up at the end and asks if you want to change anything. It's not one of these straight line programs where you make a mistake up here, and you are doomed. It's got three levels of explanation in it. One of our problems is that the contractor experiences a rather large turnover, which means that there's a lot of untrained people working the system all the time. So, what we have done here is--if you have got an old pro on your hands, the questions come real quick and terse - do this, do that, what is the value of this? But, if you've got a new person there who is not too sure about what the heck is going on, it just walks them through with very flowery explanations of what is needed by the computer at that particular time--sort of a self-training, self-documenting thing. We have three levels then, from the neophyte level, more condensed level, and a very terse level of expert.

Another key program in the system is GOVERS, short for governor. What it does is actually poke around the data base, kind of a sniffing program--sniffs around the data base to see if there's anything to do. It runs itself every 5 minutes. So, it runs day and night, constantly, goes snooping through the data base to see if there is any data that needs to go on to the next step of processing--if so, it schedules the appropriate modules.

What we found out is in order to maintain a nice quick response and the real-time quality controls, these film reading machines would have to be on their own processor. That's not such bad news, as we had a processor laying around. So this processor is more or less dedicated to maintaining a good clean feedback loop to these machines. People get very disturbed if they are reading film and they don't get immediate feedback. So we said, let's not mess around here--let's give them their own processor and get that response back there.

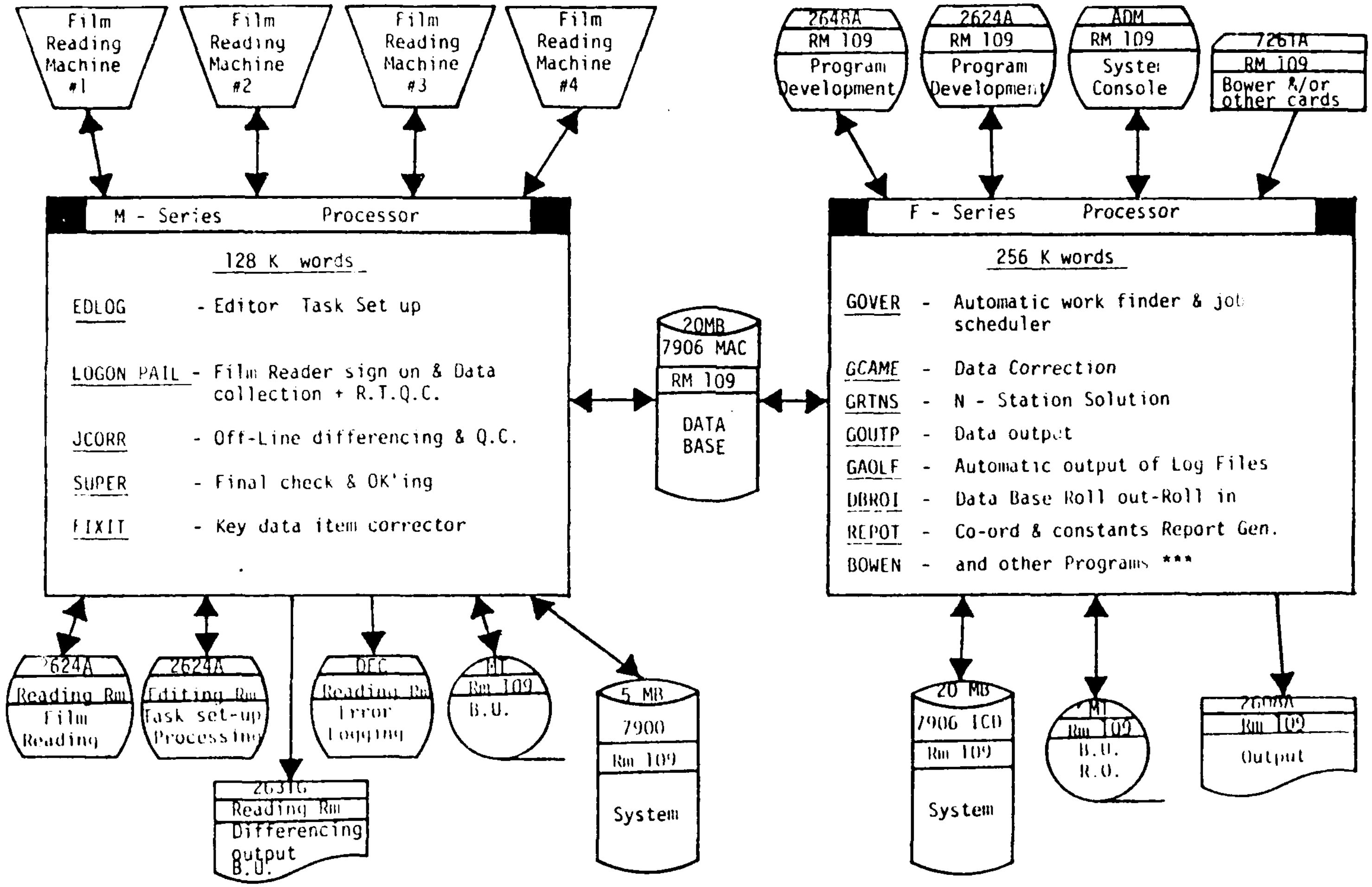
Now you've got two processors, because you've got to have another processor to run all the reduction programs on. O.K.--how are you going to get them to communicate? Well, what you probably don't want to do unless you have to is setup some D. S. loop. Vendors love to sell you these things--sell you two cards, and a big cable, and a bunch of software that doesn't work, and then your problems really begin. All the data base schema, the whole file that tells you how the data base is laid out, and what's where, is all on the disc. So, when you open the data base, you are really just looking into that schema file and seeing what's on the rest of the disc. So, all you need is to access that disc from either of your processors, and make sure that they are not both opening that data base at one time. It turns out that's easy to do in a hardware lockout because naturally, while this guy is changing that schema file, this other guy better not be puttin' and takin' as well. There is a little part that referees between the two processors and they don't really have the data base open for more than a second or two at a time anyhow. What we did was, instead of going with distributive processing, and instead of looking at any central processing unit-to-central processing unit (CPU-to-CPU) communications, to put the data base with this processor, and massage it with the other processor, and it's excellent. You don't have to worry about communications beyond that. It's essentially a mail drop, but a very, very sophisticated mail drop, because the data base software is taking care of where the actual bit of data is that you want here. For instance, if you have Test Item 123, Flight 1, Pass 1, this processor puts that data in the data base--this processor merely asks for that data generically--it says: "Give me the angle data you have for this line 123 Flight One Pass 1". That's all he has to know is the existence of that pass and ask for it by name.

If I had to summarize the design concepts which proved useful in this project, they might go:

1. Understand the data flow through the system and define data base schema around that flow.
2. Be aggressive in the specification of man/machine and Q.C.-type functions. You can be looser concerning algorithmic-type software.
3. Stabilize hardware configuration as soon as possible.
4. Don't plan any vacations.

HP-1000 Two Processor System Design

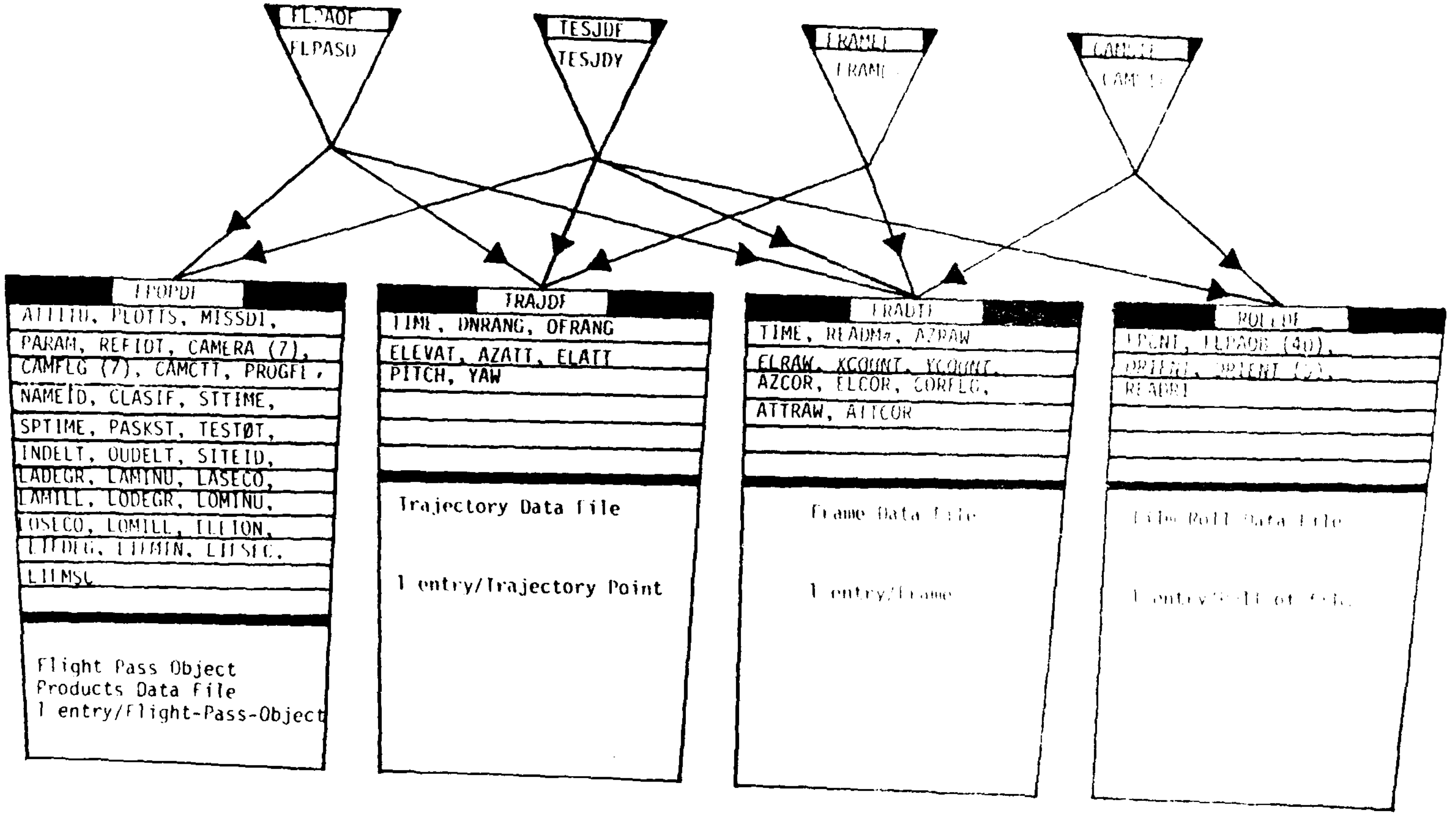
Fig. 1


Film Reading Machine #1
Film Reading Machine #2
Film Reading Machine #3
Film Reading Machine #4
2648A RM 109 Program Development
2624A RM 109 Program Development
ADM RM 109 System Console
7261A RM 109 Bower &/or other cards
M - Series Processor
128 K words
EDLOG - Editor Task Set up
LOGON PAIL - Film Reader sign on & Data collection + R.T.Q.C.
JCORR - Off-Line differencing & Q.C.
SUPER - Final check & OK'ing
FIXIT - Key data item corrector
20MB 7906 MAC RM 109 DATA BASE
F - Series Processor
256 K words
GOVER - Automatic work finder & job scheduler
GCAME - Data Correction
GRTNS - N - Station Solution
GOUTP - Data output
GAOLF - Automatic output of Log Files
DBROI - Data Base Roll out-Roll in
REPOT - Co-ord & constants Report Gen.
BOWEN - and other Programs ***
2624A Reading Rm Film Reading
2624A Editing Rm Task set-up Processing
2631G Reading Rm Differencing output B.U.
DEC Reading Rm Error Logging
MT Rm 109 B.U.
5 MB 7900 Rm 109 System
20 MB 7906 ICD Rm 109 System
MT Rm 109 B.U. R.O.
2608A Rm 109 Output

Notes: *** These programs are not related to the Data Base
B.U. = Back-up
R.O. = Roll out

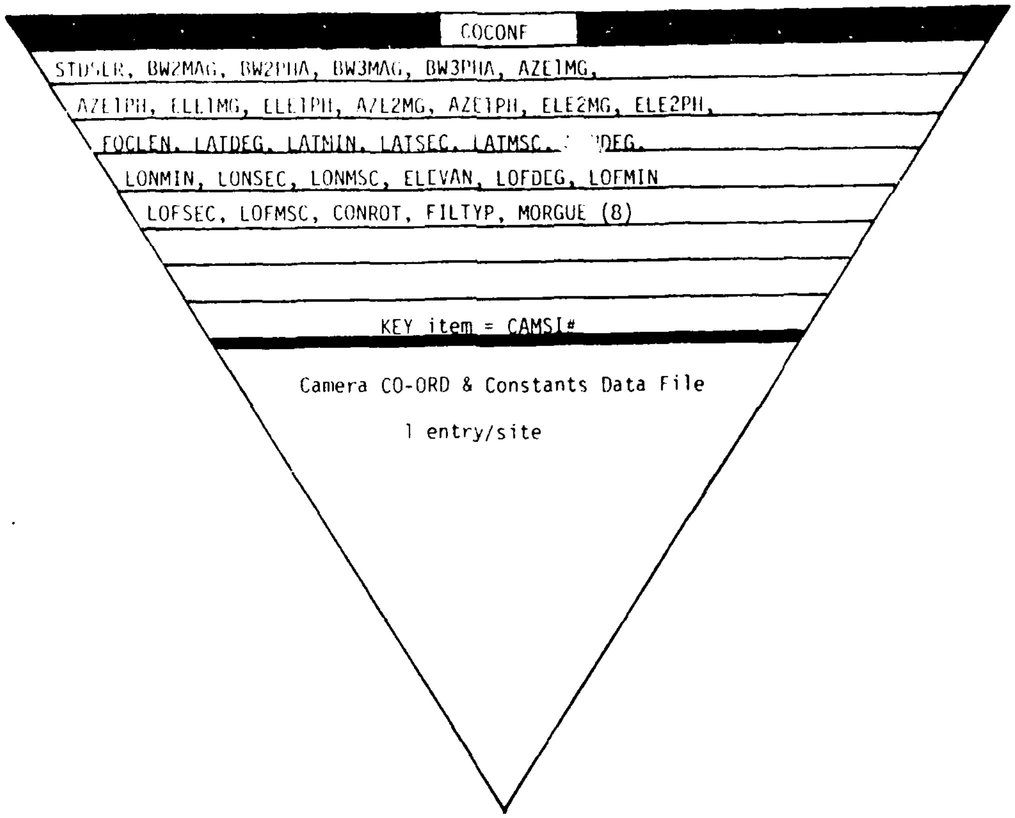
HP - 1000 DATA BASE SCHEMA ([illegible]

Automatic Master Data Sets ▽ and Detail Data Sets ⏢

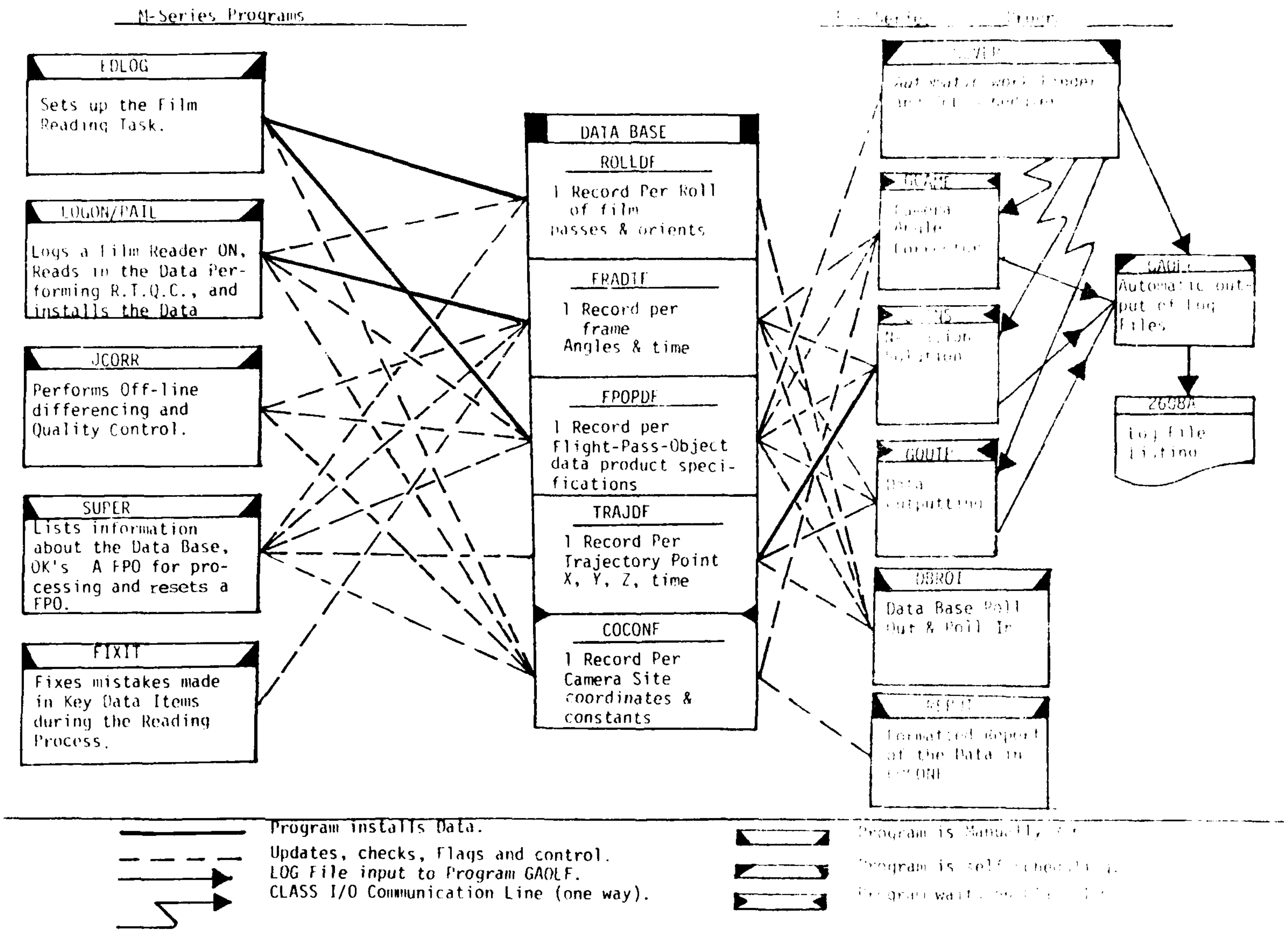


HP - 1000 DATA BASE SCHEMA (TRACAM: : 10) Part 2 Fig. 2

Camera Co-ordinates and Constants Manual Master Data Set


COCONF
STDSLR, BW2MAG, BW2PHA, BW3MAG, BW3PHA, AZE1MG,
AZE1PH, ELE1MG, ELE1PH, AZL2MG, AZE1PH, ELE2MG, ELE2PH,
FOCLEN, LATDEG, LATMIN, LATSEC, LATMSC, [illegible]DEG,
LONMIN, LONSEC, LONMSC, ELEVAN, LOFDEG, LOFMIN
LOFSEC, LOFMSC, CONROT, FILTYP, MORGUE (8)
KEY item = CAMSI#
Camera CO-ORD & Constants Data File
1 entry/site

HP-1000 Programs and Data Base Relation [illegible]


M-Series Programs
EDLOG
Sets up the Film Reading Task.
LOGON/PAIL
Logs a Film Reader ON, Reads in the Data Performing R.T.Q.C., and installs the Data
JCORR
Performs Off-line differencing and Quality Control.
SUPER
Lists information about the Data Base, OK's A FPO for processing and resets a FPO.
FIXIT
Fixes mistakes made in Key Data Items during the Reading Process.
DATA BASE
ROLLDF
1 Record Per Roll of film passes & orients
FRADIF
1 Record per frame Angles & time
FPOPDF
1 Record per Flight-Pass-Object data product specifications
TRAJDF
1 Record Per Trajectory Point X, Y, Z, time
COCONF
1 Record Per Camera Site coordinates & constants
Data
Data Base Roll Out & Roll In
DBROI
Formatted Report of the Data in COCONF
GAOLF
Automatic output of log files
2608A
log file listing
Program installs Data.
Updates, checks, Flags and control.
LOG File input to Program GAOLF.
CLASS I/O Communication Line (one way).
Program is Manually
Program is self scheduling.
Program waits

SOFTWARE MAINTENANCE - COMING OUT OF THE CLOSET

PRESENTOR: W.J. EGAN
HEAD, SYSTEMS ENGINEERING BRANCH, CODE 1221
PACIFIC MISSILE TEST CENTER

ABSTRACT

This presentation traces the evolution of tactical software maintenance policies, practices, and procedures at the Pacific Missile Test Center from 1970 to the present and postulates probable trends for the coming decade based on this experience. The Center's initial involvement in tactical software projects will be presented along with the problems, insights, and solutions which evolved as a result of the day-to-day working environment. Administrative and managerial solutions gained as a result of this corporate experience, and tested by time, will be examined and a prognosis for the future, as viewed from the practical work center, will be advanced. Several contributions of the Pacific Missile Test Center to the field of software maintenance and management, such as the use of integrated test sites, software change control, and life-cycle planning methodologies, will be discussed, as well as PACMISTESTCEN's role in influencing key NAVAIRSYSCOM and DOD instructions on software management planning.

Maintenance of tactical embedded computer software has often been treated as a second-class relation to new system development. Maintenance represents the product of past romances; new systems stimulate us and foster our hopes and dreams. Consequently, and understandably, new systems bathe in the glow of the limelight while maintenance of developed systems retreat to some darkened closet. Things may well be changing. Assessing software management problems in the Department of Defense back in 1978, De Roze and Nyman state:

> "The distribution of software costs for all military systems for a given fiscal year shows that 68 percent of known costs are consumed in development of New Systems (R&D), while the remaining 32 percent of the known cost is categorized as operation and maintenance (O&M) of systems already in the field. The majority of complex software systems are new and still in the development cycle. As these new systems are deployed, this cost distribution will reverse to emphasize the increased O&M burden. This, coupled with greater system longevity, may ultimately result in a five or ten-to-one ratio of O&M cost to R&D cost when viewed over the total life cycle of a typical system. With these projections we will need an Army of software maintenance engineers". (p. 310)

These thoughts, written 4 years ago, suggest that when software maintenance costs achieve ratios of five-to-one, or better, over software development costs, then it will become increasingly difficult to keep such a function relegated to some restricted closet. Throughout the Naval Air Systems Command (NAVAIR) in general, and at the Pacific Missile Test Center, software maintenance of tactical embedded computer systems has started to emerge from their closets.

Software maintenance of operational programs involves three change categories: (a) corrective changes - dealing with failures in processing, performance, or implementation, (b) adaptive - responding to anticipated in the data or processing environments, and (c) perfective - enhancing processing efficiency, performance or system maintainability.[1] More than just terminology is at stake here. It is critical that funds allocated for maintenance (O&M) shall not be utilized for development.

[1] Mooney, J.W. "Organization Program Maintenance." Datamation 21 February 1975, 63-66.

In general, within NAVAIR, "software update" is accomplished via utilization of non-O&M funds and "software repair" with O&M funds. Software update "results in a changed functional specification" and software repair "leaves the functional specification intact."[2] NAVAIR software policy does permit minor perfective-type software modifications with O&M funds as long as they are accomplished in concert with implementation of corrective/adaptive change actions[3].

Software maintenance can be viewed in three distinct phases or eras -- the very early era, at least 10 to 12 years ago, started as a period of "mismanagement." Currently, we have evolved into a period of "mirco-management," and as we proceed forward into the future, a period that must surely come, we shall enter an era of software maintenance "macro-management."

Phase One - Software Maintenance Mismanagement

Looking back at where we came from, in the airborne arm of the Navy, we had a very unique problem to solve. The early 1970's were a somewhat unsettling time in the United States. It was a time when there was much national agonizing over the war in Vietnam, R&D budgets were declining, and DOD was undergoing transformation as a result of the President's blue ribbon commission. There were changes in DOD and in decision-making processes and testing. At this point, systems engineering/system integrations were considered to be an art.

Projects within NAVAIR were developing requiring software support/ software maintenance and some very minor funding became available. These miniscule funds were allocated to NAVAIR field activities that were willing to cough up a few wayward engineers that knew a little about software. Space was found, at times no more than temporary trailers or some undesirable converted storage structure, but it was at least something. We said to them - "we want you to do software maintenance." That was just about all there was to beginning; no more profound than establishing a closet here and there and asking people to go to work. (As illustrated in Figure 1.)

[2] Boehm, B.W. "The High Cost of Software," Proceedings of Symposium on the High Cost of Software, Monterey, California, 1973, 27-40.

[3] NAVAIR Software Management Advisory Committee, "Naval Air Systems Command Software Management Manual," November 1981.

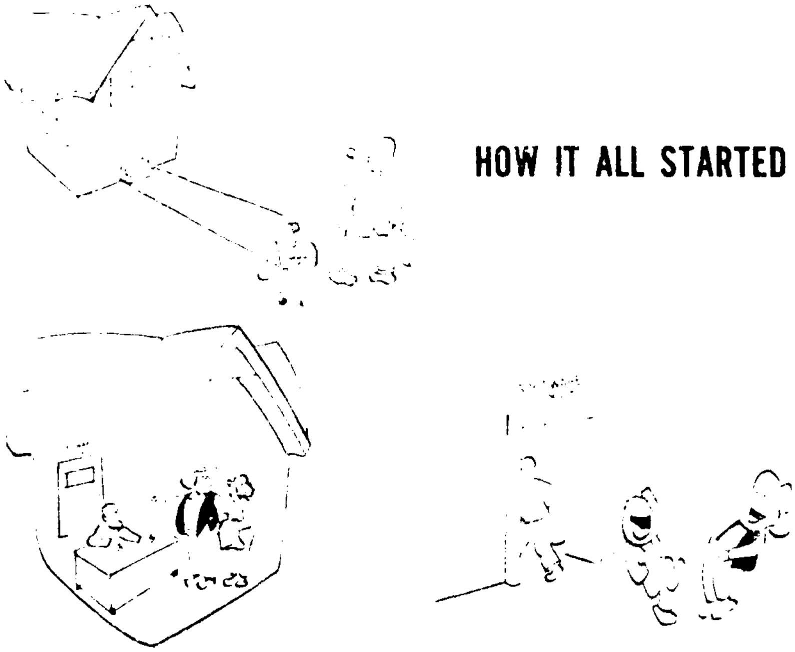


Figure 1. How It All Started.

In those early days as it is today, much attention was focused on the management aspects of software maintenance and the management end of things. There is quite a hierarchial structure of systems, whereas software/hardware people may take a myopic viewpoint of these systems and not see the total system in the proper context. The Navy did not assert itself and face up to some of these problems. Even today, with as great a capability as software possesses, and weapon systems provide, software maintenance remains, to a large extent, in the "closet."

The Pacific Missile Test Center received an assignment in 1973 to serve as the Software Support Activity (SSA) for the F-14 Weapon System, one of the first systems for which a software support activity was assigned. The F-14 is a supersonic fighter that can carry infrared SIDEWINDER, PHOENIX and SPARROW missiles and a 20-millimeter gun. It also is equipped with a state-of-the-art fire control system backed up by four onboard computers. It is a fast, lethal, and highly sophisticated aircraft. It is a system that will not tolerate randomness, errors, or faults of any kind throughout its life cycle because the weapon system cannot perform its function without the computer and software. The early period of software maintenance development was characterized by this exposure of a full-fledged, very comprehensive weapon system.

Figure 2 depicts the F-14 weapon system development and initial deployment posture. The left-hand column identifies testing and integration milestones while the right-column addresses initial deployments through 1980. As can be seen, there is some overlap of development and deployment. This constitutes a real problem for any testing/integration program because of the need for Initial Operational Capability (IOC) in a complete and timely manner. The IOC nearly always, for a variety of reasons, remains nonslippable. They wanted two F-14 squadrons onboard the *Enterprise* on Labor Day 1974 and they had to be there. The message here is - having gone through all that testing/integration program, the system development finds itself right on top, or even past that period appointed for deployment. Realistically, we still have many system problems that now befall the maintenance guy, the guy who must now be immediately capable, as the software maintenance activity. Early involvement of this activity in the development process is therefore crucial. Figure 2 covers most of the traditional and classical events that take place with all major weapon systems over the years. Unfortunately, with this system, it really was not until fleet delivery that we realized the total scope and magnitude of the effort involved in supporting the software with this type of system.

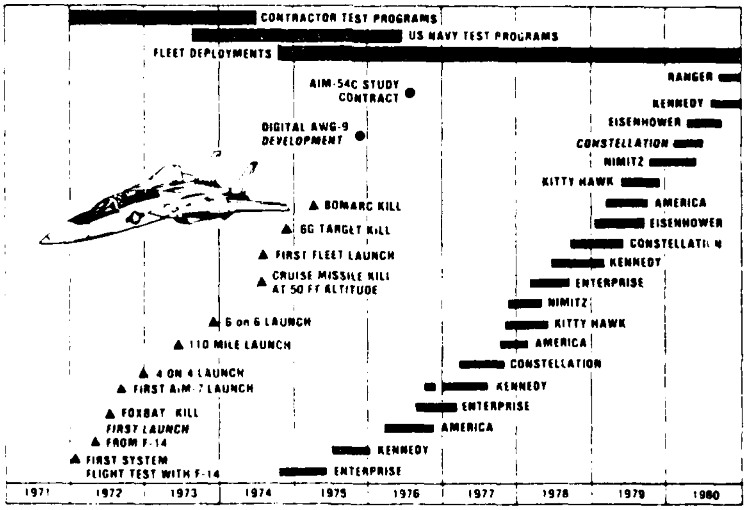


Figure 2. F-14 Weapon System Development.

Figure 3 illustrates the process that evolved to accommodate the software life cycle. There is nothing really unique about it and all the aspects are readily understandable. However, during prosecution, one thing became very clear; as system development evolved, many problem's fell on the activity that had responsibility for the maintenance. We realized that the maintenance activity was doing more of other people's jobs because overlooked biases remained in the system throughout the process. Problems that fell through, fell all the way down to the maintenance activity. (A new trickle-down theory.) The maintenance activity rapidly came to the realization that -- here we are, supposedly on the trailing edge, supposedly responsible only for maintenance and actually finding ourselves with the added responsibility of having to clean up other activities' work. In spite of this, the record shows that a significant amount of added value was provided to the weapon system. That remains true today, because when we talk about software maintenance in the Navy we are not simply discussing fixing a few bits here and there. Instead, we are talking about establishing a capability that allows provision of added value via the flexibility and power of software.

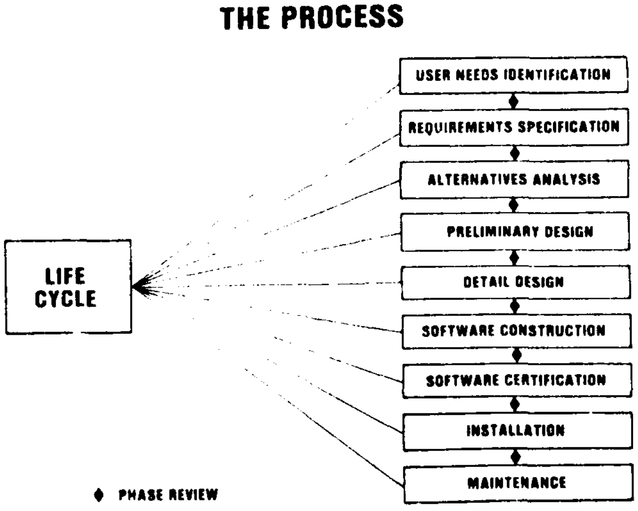


Figure 3. The Process.

Software maintenance requires special tools and laboratories. In the case of the F-14 a facility has been established that is one of the finest laboratories in the DOD. Early thoughtful planning did not provide this, it was instead an outgrowth of the NAVAIR decision that the F-14 weapon system integration should be accomplished at a Navy facility. The Systems Integration Test Station (SITS) replicates the actual prime hardware and software both in configuration and physical location. Through simulation and stimulation techniques utilizing "real world" operational scenarios, SITS significantly reduced the time to determine system effectiveness against various threats. The other real benefit of this system is in terms of dollars, etc. It costs approximately $1,000 an hour to run the lab. An instrumented F-14 utilizing range facilities and data reduction cost approximately $30,000 per flight. A 30 to 1 savings in cost is realized by using this laboratory.

This is how it works. An F-14 can be sent up on the range. The exact same data reduction facilities are present in both SITS and in the airborne F-14. They can be given the same data at the same time and the results can be compared. It was, and remains, a development tool, proven to be every bit as effective during the maintenance phase. It is an extremely valuable and capable tool from which we get sufficient data to make correct management decisions.

The mismanagement phase of software maintenance has provided the four following significant lessons:

Early Involvement - If there is to be a software support activity, responsible for the maintenance of the weapon system software, it must be involved very early in the program. If the software support activity is not involved in the writing of the request for proposal it is already too late and behind the power curve.

Communications - Effective constant communications between system management and the field activity is paramount. Lines of communication between the software support activity and the fleet user are critical.

Operator Involvement - One success in the F-14 program has been the ability to get our fleet users intimately involved with the change/decision process. User communications on the F-14 program involve meetings of operation advisor groups that bring actual pilots and naval flight officers in to talk with the engineers. There is a maintenance advisory group that brings in maintenance personnel from the various squadrons to review problems with the designers.

Decision Process - The decision making process should not be something that is too complex. Instead, it has to be something that is responsible, and meets the stated needs. This issue will be discussed later in this paper.

Phase II - Software Maintenance "Micro Management"

Where are we now and where are we heading? Phasing into the micro-management period presents no real line of demarcation from the mis-management period. We have realized, as illustrated in Figure 4 that over the years the "closet" has grown, although the actual size of the landlord structure remains the same. More and more workload with increasing responsibility migrated to the occupants of the software closet. Quietly, growth took place, for the most part, unknown outside the apparently confined structure. Additional workload and attendant responsibility did not take place without some penalty. In this case a big new system management problem to solve.

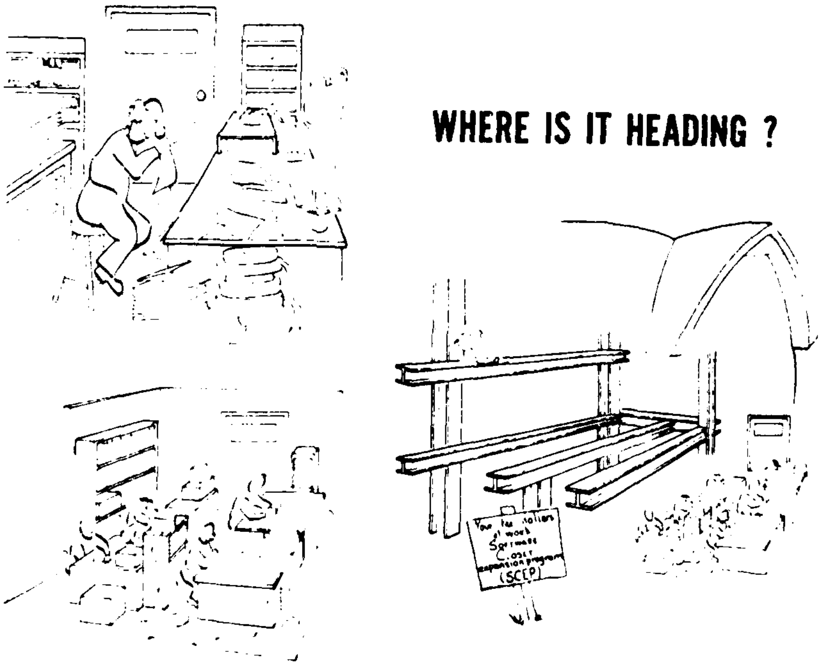


Figure 4. Where is it Heading?

Figure 5 illustrates the situation that now confronts software maintenance. The various components of the problem compare with some big jigsaw puzzle thrown on the floor and exploded into various pieces requiring a not-so-simple reassembly. Not having good perceptions of sizing, and good perceptions of interaction, complicated the fundamental recognition of the solution. Solving the puzzle required that we concentrate on some of the cardinal elements that were needed to deliver quality working products to the fleet on schedule. The real challenge was in pulling all the diverse software management components together into a cohesive whole.

Software problems were not the only concerns, as many problems had to be aodressed that were not traditionally categorized in the software area. Piecing those together and funding integration tools was a challenge. At a stage when the system is deployed, money for elements such as research and development and maintenance gets so scarce that the issue of doing software maintenance within budget became a difficult problem.

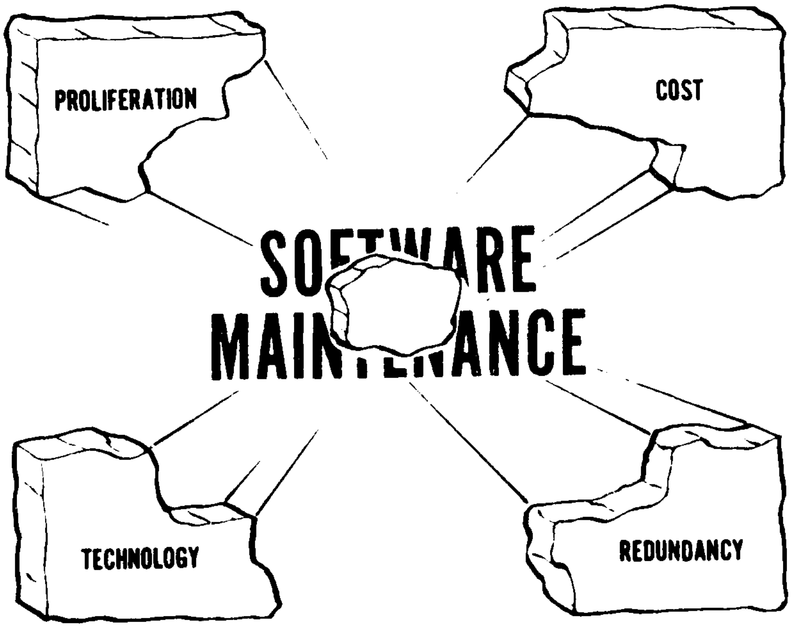


Figure 5. The Fragmentation of Software Maintenance.

Redundancies occur when varied companies are working with a government industry team. Grumman, Hughes Aircraft, General Dynamics, Philco/Ford, and Raytheon, all had their perceptions of the integration problem from a different perspective and it was difficult to bring them together. Drawing on all sorts of disciplinary technology to make software maintenance a reality, it was recognized that it was different from the research, development, test and evaluation process and from the conventional practices for maintaining hardware.

Recreation of the jigsaw puzzle involves consideration of at least four fundamental elements. Figure 6 illustrates these key elements of adequate software support. They include: (a) dedicated software support activities, (b) dedicated integrated laboratory systems, (c) effective software life-cycle planning and management, and (d) structured closed-loop change processing with user involvement.

These are all different in that they do not correlate with anything we have done in the past. First we had to dedicate the resources and the activities toward this vitally important function of Fleet support. Dedicating resources continue to be a problem today because as more and more resources become dedicated to the software area (with all of government shrinking in the number of personnel) it forces other areas to compress at an even higher rate. This situation results in all kinds of trouble for the growth areas.

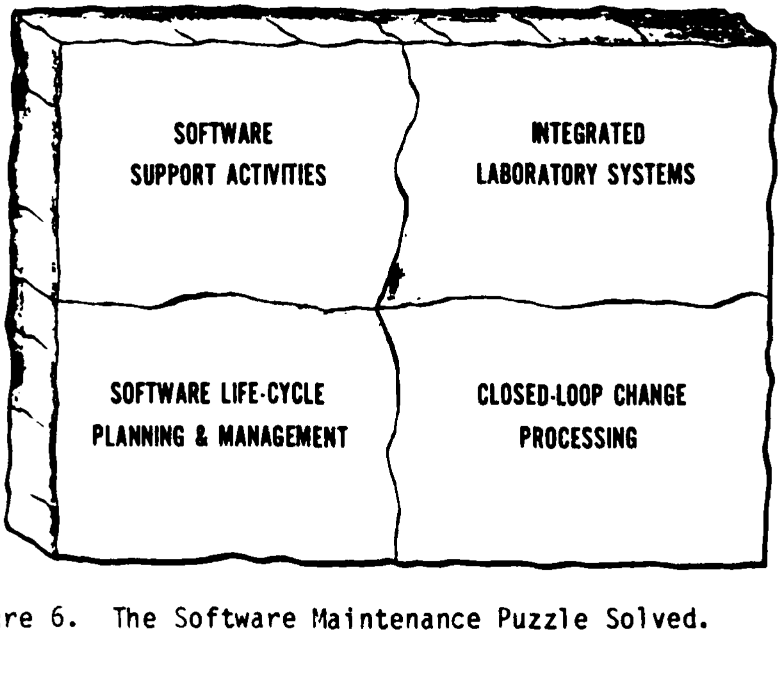


Figure 6. The Software Maintenance Puzzle Solved.

The sum total of this process developed an issue confronted with more criticism than it deserves. The F-14, from a mission performance point of view, has provided added values through software maintenance without grounding or rotating the F-14 in the fleet. Over the past decade, costs went up, the resource requirements went up, and the criticism of the methods of doing business went up. A solution must be sought as our systems become more and more software intensive.

Phase III - Software Maintenance Macro Management

The issue of software economics has to be integrated somewhere from micro-systems management to macro-systems management and this brings us to a very important crossroad. Having developed some unique disciplines, and having a good record of product deliverables we have proved that software maintenance does work. Now, rather than building from "closets" and allowing complacency to lead to obsolence we should seek the blending of innovative ideas with the reality of today's software maintenance concepts and continue to bring new life to this multifaceted area.

Should we permit software support to become "microized" as just a subset of software maintenance and not address the bigger problems, not look at systems engineering, not look at the management disciplines that we need to apply, we will never reach that "crystal city" or that "century city" shown on the right of Figure 7. Software maintenance might well instead find itself mired in some kind of "software appalachia," particularly considering the economic posture that we are in today.

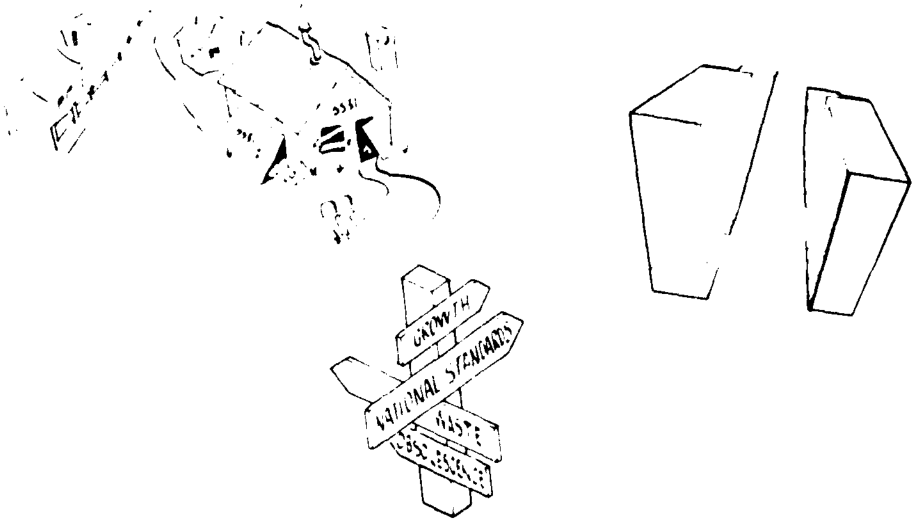


Figure 7. Software Maintenance at the Crossroads.

Product obsolence must be taken into account when we ask ourselves "Where Do We Go From Here?" No matter what product line one is in, there is always a life cycle to it. We are entering into an era of software where the product is maturing via software maintenance. and we should not lose sight of where that leads. In an article on marketing written by Professor Ted Lents, he refers to an example where; if you produce and sell buggy whips, when horse buggies/horse carriages go away, you stop pushing buggy whips or you will go away.

Software maintenance is maturing. Along with software/hardware and firmware a different context is exposed. "Brainware" - people have to use their brains to figure out better ways to do the business and integrate new methods into the total system. A challenge in blending the innovation of ideas with the realism of software maintenance is to optimize the acceptability in the real world.

It is a difficult task to make sure that the conventional and traditional practices embrace the issues of software and the system as a whole. The "closet" just cannot keep on bulging. The program must take a perspective on growth, change development, architecture systems and management. From a systems viewpoint, we are emerging into a maturing industry and one that is critically important, not only to the United States and its defense industry, but also to man.

POINT OF VIEW

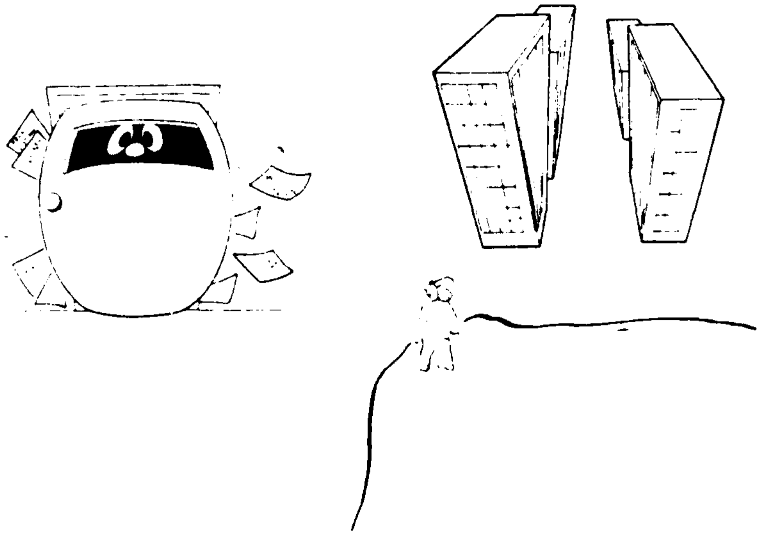


Figure 8. Point of View.

In the 1980's, we are part of a growing information and knowledgeable industry. Our nation's strength will depend on the knowledqe and know-how in hardware, software, firmware and brainware. We have to propogate more system disciplines, more system engineering and better systems management, both in macro and micro aspects. If we are to be a reliable industry in the future, we must be proficient in both hardware and software.

Can we excel and bring SOFTWARE MAINTENANCE OUT OF THE CLOSET?

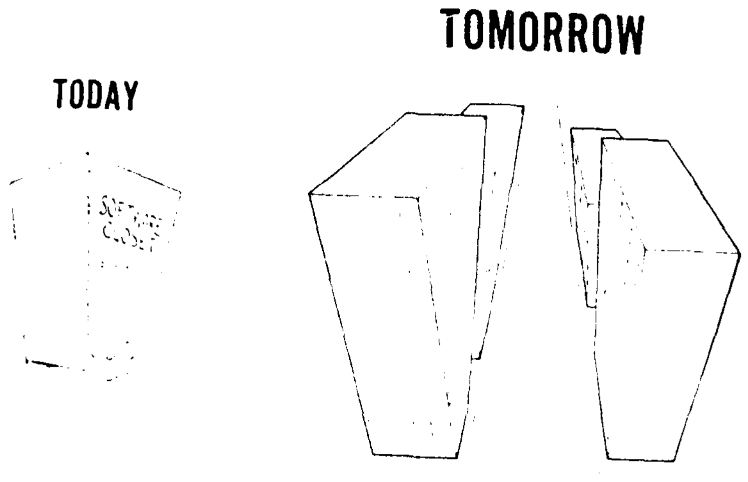


Figure 9. Today and Tomorrow.

For additional information, contact:

W. J. Egan
Head, System Engineering Branch
Code 1221
Pacific Missile Test Center
Point Mugu, CA 93042

THE ANATOMY OF A SUCCESSFUL PROJECT
OR
SUCCESS THROUGH COMPOSITE STRUCTURED DESIGN

By

W. J. Kirklin, Manager
Central Computer Complex
RCA Company
ESMC, Patrick AFB, FL

This paper discusses the techniques used to maintain a tight schedule for a major programming project with limited manpower. Major tools used were: composite structured design, structured programming, and a project team resembling IBM project team concept.

Preceding page blank

INTRODUCTION

It is impossible to write a good program with a bad design.
It is difficult to write a bad program with a good design.
It is cheaper to do it right.
It is faster to do it right.

These four statements summarize the attitude of the team throughout the project. The report that follows shows that, at least for this project, the attitude was justified.

I. The Project

The subject project was to design and implement an interactive system to:

a. Support ETR Range Scheduling operations and

b. Provide accurate and accountable resource utilization hours in support of direct cost reimbursement using a unit service charge concept.

ETR was committed to implement reimbursement under unit service charge at the start of FY 82.

Range Scheduling had an operational interactive system which contained in its data base a portion of the resource utilization data needed to meet the new requirements. No other existing system contained any of this data. The existing system required expansion to meet expanded scheduling needs, including the ability to include classified operations in its data base. Initial analysis showed that it would require more effort to modify the old system to meet the new needs than it would to design and implement a new system.

The old scheduling support system, the Mechanized Range Scheduling System (MRSS), maintained a data base of seven files:

1 Schedule File
2 History File
3 OD/Annex File
4 Resource File
5 Test Number File
6 Support Card Validation File
7 Resource Utilization File

MRSS consisted of 13 CDC CYBER 74 programs which maintained the data base; outputted daily, weekly, monthly and other internal reports as well as on-demand reports; and assisted the scheduling operation in its daily activities.

There was considerable redundancy in the MRSS data base.

MRSS was not as responsive to the schedulers needs as it should have been.

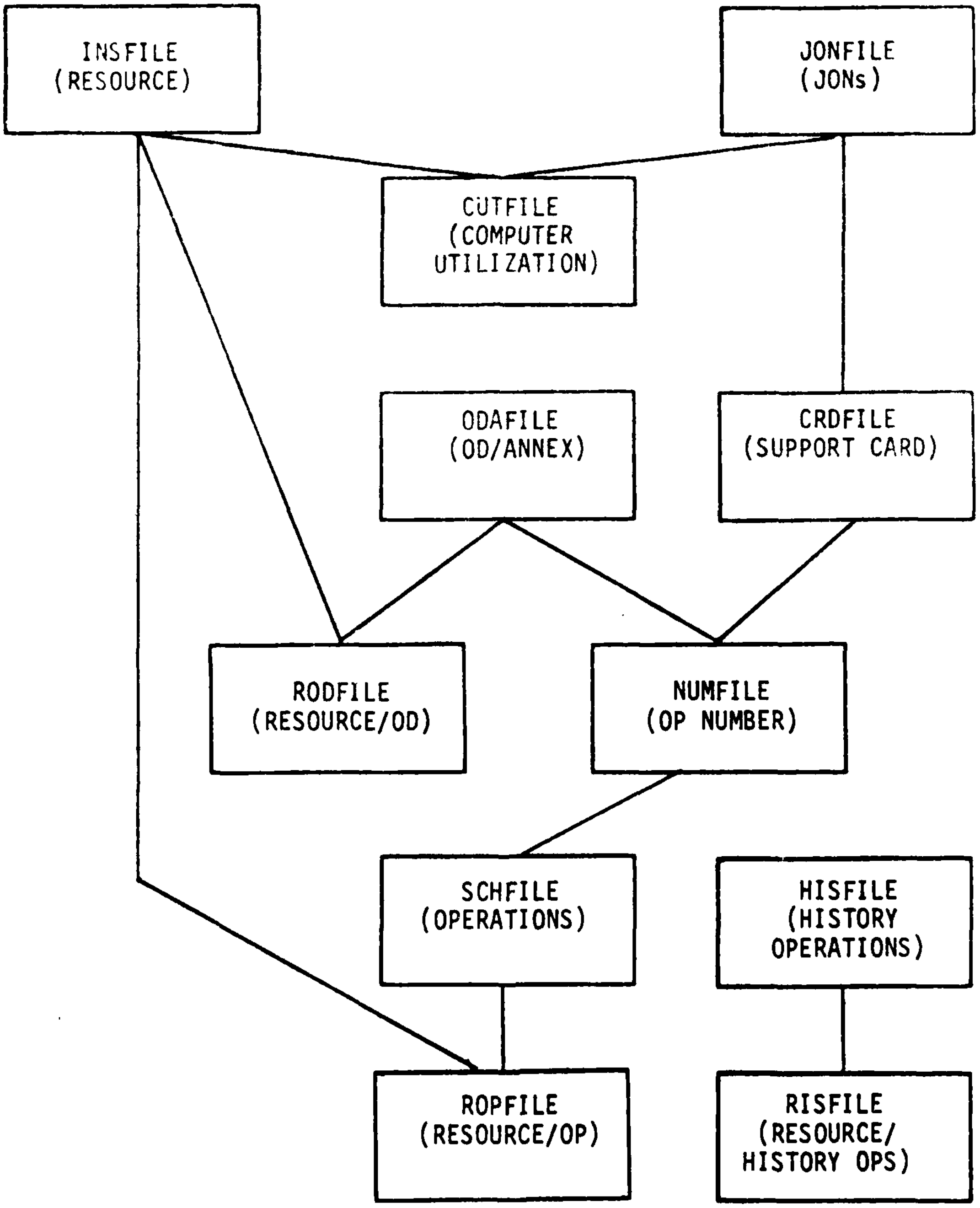
The new system, the Mechanized Resource Management System (MRMS), was to use the same data base as MRSS with some relatively minor expansions Where MRSS was a collection of independent programs MRMS was to be a single program subdivided into ten independent sections processing the various input types:

1 Job Order Number Commands
2 Support Card Commands
3 Resource Commands
4 OD Commands
5 Number Commands
6 Operations Commands
7 Schedule Commands
8 Conflict/Commitment Commands
9 File Commands
10 Miscellaneous Commands

The first six of these sections are used to maintain the data base. These commands are used to add to, delete from, or modify the contents of the various files in the data base. Sections 7 and 8 produce special reports required for the normal operations of Range Scheduling. Section 9 controls the handling of any print files created by the program. Section 10 contains a set of small commands which control various program execution parameters (e.g., default print files, schedule/history mode, etc.).

There was also to be a section to produce the numerous standard reports produced by the system.

MRMS DATA BASE OVERVIEW



II. The Tools

A. Structured Programming

The time was right for structured programming. L. M. Holland had recently presented a 3-day in-house course on structured programming to many of our RCA analysts/programmers. CDC had just released its FORTRAN V. The literature was saturated with the virtues and vices of structured techniques. (The virtues appeared to be clear cut winners over the "sour grapes" vices.) We clearly needed all the positive features promised by structured techniques, so this project was selected as a pilot project for structured programming. While it is generally risky to try new things on a major project, the potential benefits far outweighed the risk factor.

B. FORTRAN V

The use of "structured FORTRAN" was dictated by the decision to use structured programming techniques. This introduced three negative considerations:

1. A new language and the necessary learning time on the part of the programmers.
2. Existing software was not compatible and numerous library routines had to be rewritten in FORTRAN V so they could be used for this project.
3. A new release is frequently full of hidden errors.

The team selected for this project tended to minimize the risks. Both programmer analysts were new to CDC FORTRAN and, therefore, learning FORTRAN V would be only slightly more difficult than learning CDC-unique features in FORTRAN IV. Also, both were recent graduates and had been exposed to some structured techniques in their schooling. The lead analyst was sympathetic with structured techniques.

The software incompatibility was a cost of progress. Sooner or later we were going to go to FORTRAN V. Why delay?

The risk of software bugs was considered acceptable (CDC had 16 months before implementation to get the bugs exterminated). Actually these bugs did cause some minor problems as seen in Section IV.

C. Structured Design

This is the most significant tool used. Without structured design MRMS would still be floundering in a prolonged debug phase and I would be making up excuses for a failure instead of presenting a paper on a success.

It is difficult to write a bad program from a good design.

The MRMS team is unanimous in its praise of Glenford Meyer's "Composite Structured Design" as a tool for good design. In fact, this particular tool was selected by the team and not suggested by management. Surely the risks associated with any unknown untried technique exist, but this team was so enthusiastic about this choice that they refused to recognize any risk.

D. DMS-170 (Query/Update)

DMS-170 is CDC's data base management system. This system was purchased for us in the very early days of this project. The team viewed DMS-170 with very mixed feelings. This was a totally new tool to us and as such was a very intriguing toy. But it was also a threat. We felt confident we could meet our due date using our plan. Query offered a tempting promise for a short-cut but one that might be booby-trapped. Query was designed so it could be used by a naive user--does it have sufficient power to meet MRMS requirements? If it has sufficient power to meet the need, does it also have sufficient power to allow our naive user to destroy his data base? Did we have time to explore this tool and still have time to complete our planned approach should DMS-170 be either too powerful or not powerful enough? The only choice we felt comfortable with was to proceed with our plan and investigate DMS-170 in time available with an eye to future applications.

E. Algorithm Statement [Pseudo Code Program Definition Language (PDL)] Instead of Flow Charts

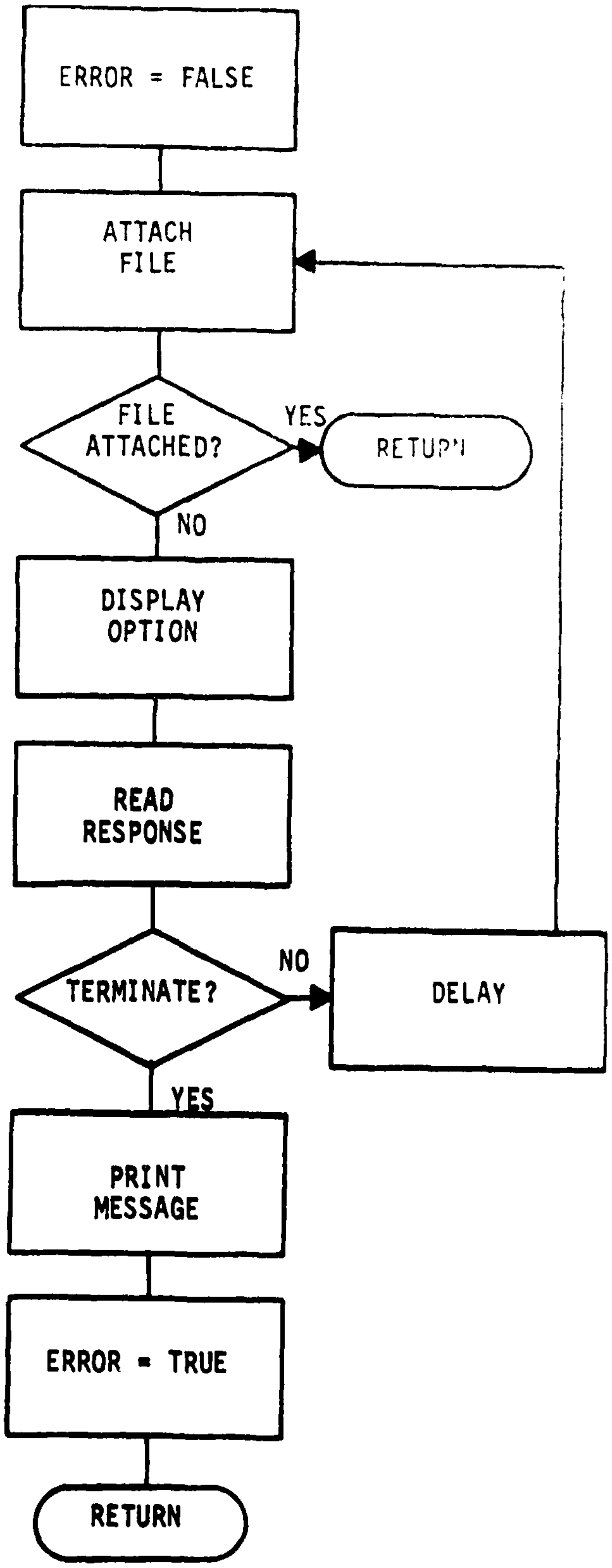
Flow charting is a tried and true tool--why abandon it on a critical project? It is also an abused tool. Few flow charts are worth the effort taken to create them. Flow charts are tedious to draw and both difficult and costly to revise. An algorithm description is easy to write and easy to revise. The algorithm description provides a better insight into program structure than flow charts. This is especially true when the algorithm is written in a pseudo-FORTRAN V.

The following is an example of the algorithm specification used during the design phase. The algorithm specifies the steps to be taken when "attaching" a data base file to the program. If a file cannot be "attached," the terminal operator is given the option of making another try or terminating the procedure.

The corresponding flow chart is also shown.

F. ALGORITHM

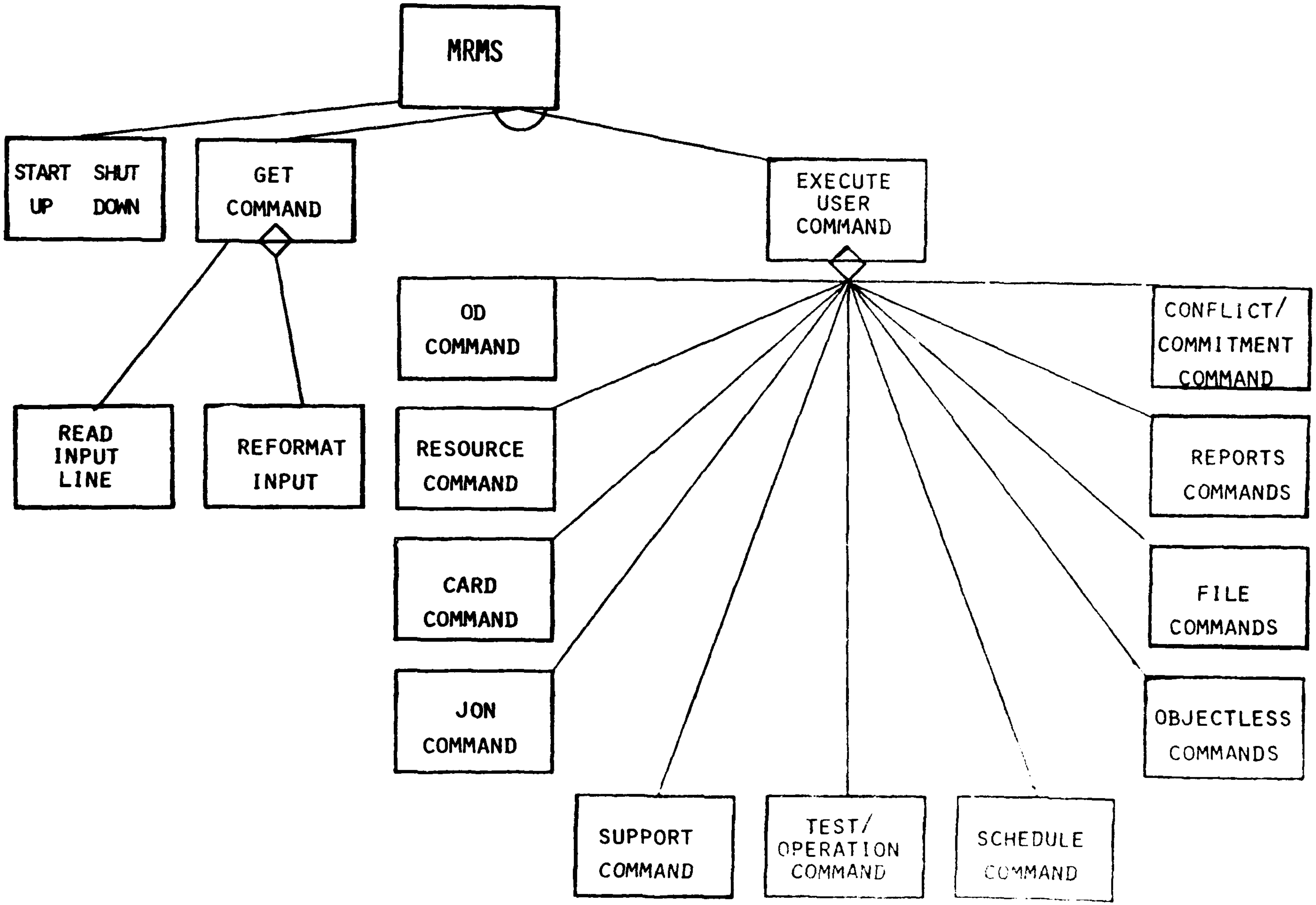
```
ERROR ← False
Loop:
   Attempt ATTACH
   Exitif (attach good)
   Give operator option to terminate
   Erase prompt
   If (terminated) then
      Print terminate message
      ERROR ← True
      Exit loop
   Endif
   Wait before next try
End Loop
Return
```



F. Structure Charts

Structure charts are a subtool of structured design. A structure chart shows how each program module is related to each other module.

MRMS SYSTEM DESIGN



G. Interface Charts

Interface charts detail the way data is passed between modules. Use of these charts force an early definition of variables and tends to stabilize design.

Page 1
Date 22/1/81

Interface Chart

Program MRMS

Inter-face #	Calling Module	Called Module	IN	OUT	Coup-ling	Type of Call	Freq of Call
1	MRMS	UP	PGM	–	Data	N	[illegible]
2	"	DOWN	–	–		N	[illegible]
3	"	GETCMD	–	LINE, NTOK, TOKEN	"	I	[illegible]
4	"	MRMX	LINE, TOKEN	–	"	I	[illegible]
5	GETCMD	READLN	–	LINE, ENDFILE	"	N	[illegible]
6	"	SCAN	LINE	NTOK, TOKEN	"	C	91
7	SCAN	GNSEP	LINE, CURSOR	NEXT, SEP	"	I	[illegible]
8	"	NXTOKN	LINE, CURSOR	CURSOR, ITOKEN, JTOKEN, KTOKEN		I	[illegible]
9	"	BUILT	ITOKEN, KTOKEN TOKEN, NTOK, JTOKEN, LEVEL	fun, NTOK	"	I	[illegible]
10	"	LYNX	TOKEN, NTOK	TOKEN	"	C	90
11	"	TRIM	LINE	fun	"	N	100
12	NXTOKN	LEXICAL	LINE, ITOKEN, STATES, CHARS	CURSOR, KTOKEN	"	N	100
13	LYNX	BLINK	TOKEN, LAST, IT	fun, TOKEN	"	I	[illegible]
14	"	SLINK	TOKEN, IT	TOKEN	"	I	[illegible]
15	"	LEV	TOKEN, IT	–	"	I	[illegible]
16	READLN	LOGNTRY	–	–		N	[illegible]
17		LOGTIME	–	–		N	91
18		LOGTIME	–	–		N	1
19							

H Structured Walk Through

Structured walk throughs maintain team identity and keep team members on the same track and fully informed on what each team member is doing.

III. Philosophy and Plan

A. Priorities

1. The first and overriding priority for this project was to finish on time. Until the last few months the ETR was committed to implementation of direct cost reimbursement by unit service costs on October 1, 1981. Until this mandatory date was slipped for a year the implementation date was mandatory.
2. Implement all planned features of the program by the due date.
3. Be responsive to the user's needs.
4. Provide a maintainable program. This priority relates very closely to the decision to use structured techniques.
5. The fifth priority was efficient use of storage. Elimination of redundancy within the data base also enhances long term stability of the system by eliminating the possibility of divergence of data in the redundant sections. Divergence will eventually occur when data in one file is updated and another file is not--perhaps through system failure.
6. Code in a single consistent style. This last priority is of little import to the value of the system but is rather a symptom of the determination of the team to show that it can be done.

B. Task Sequencing

1. The first task for the team was to write its own Project Standards Manual. This may appear paradoxical as this task relates to the last item on the priority list, but this task has hidden values. The act of cooperatively defining a style and consistent mode of approach establishes a team identity; it removes the tiny decisions of styles from the production phase; it promotes compatibility in form which increases usefulness of utility type subroutines; and it tends to cement a team spirit with the challenge of making a program written by two or more programmers appear as if it had been done by one programmer. This is another facet of the "egoless programming" that has appeared so much in data processing literature.

2. Having established a team identity the next task was to design the data base. This must be the first stage of the design because the data base design drives all other phases of the design effort. It is impossible to design a program to maintain a data base that has not been defined.

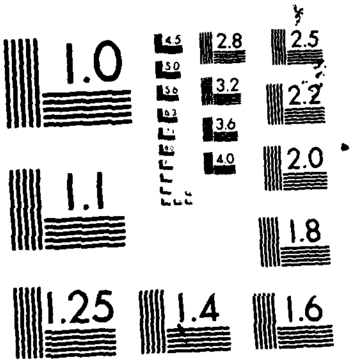
3. The next task was to design the man/machine interface. This task is to define the language that the scheduler will use to perform his tasks with the computer.

4. Having defined the language the next task is to document the language by writing the User's Manual. Having the User's Manual written at an early stage in the project allows the user to pre-examine the product to see if he understands and if he approves of the product. Hopefully, it opens a dialogue between programmer and user which will continue throughout the project.

AD-A117 995 SOFTWARE ENGINEERING SEMINAR (7TH) 28 APRIL - 2 MAY 1980 ELGIN AIR FORCE BASE FLORIDA(U) RANGE COMMANDERS COUNCIL WHITE SANDS MISSILE RANGE NM DATA RE

2/2

UNCLASSIFIED 02 FEB 82 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

PROJECT SCHEDULE

```
           1980                        1981
ACTIVITY   JUN JUL AUG SEP OCT NOV DEC JAN FEB MAR APR MAY JUN JUL AUG SEP OCT
           +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
  A.1      ****************************************************************
  A.2      ******
  A.3            ******
  A.4                  ********
  A.5                      ******
  A.6                            ********
  A.7                                  **************************
  A.8                    **********
  A.9                              **********
  A.10                                   ******************************
  A.11                       ************
  A.12                                                       ************
  A.13                                                               ****
  A.14                                                                   ****
           +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
                   ↑   ↑     ↑ ↑ ↑                                         ↑
                 E.1 E.2    E.3 E.5                                      E.6
                              E.4     EVENTS
```

ACTIVITY

MRSS
A.1 - Production Maintenance
A.2 - Implement Security Mod

MRSS/MRMS
A.3 - Orientation and Training For Project Team

MRMS
A.4 - Design Data Base
A.5 - Design Man/Machine Interface
A.6 - Design (0,0) and Primary Overlay Modules
A.7 - Design Secondary Overlay Modules
A.8 - Pre-build: Utility and Service Routines, CCL, etc.
A.9 - Build (0,0) and Primary Overlay Modules
A.10 - Build Secondary Overlay Modules
A.11 - Create Data Base For Software Testing
A.12 - Training For User Personnel
A.13 - Acceptance Testing
A.14 - Production Maintenance

EVENTS

E.1 Programming Standards Manual
E.2 - Receive Detailed Report Formats From ROS
E.3 - Receive Data To Create Resource File From ROS
E.4 - Design Review For Data Base and Man/Machine Interface
E.5 - Users' Manual
E.6 - Certification

5. We are now ready to define the major program sections which were at least conceived during the writing of the User's Manual. In fact, from here on out we were driven by what was put in the User's Manual. High level structure charts and interface charts will now begin to appear.

For the first five steps the order is strictly definable. The remaining activities are orderable within groups but not between groups. There are two types of groups: programming and data base.

The internal order within the programming groups is:

1. Design
2. Document
3. Code
4. Test

The steps in the design process are:

1. Identify the tasks that must be performed to accomplish the purpose of the program. Each task becomes a module of the program.
2. Define an algorithm to accomplish the task of each module.
3. Organize the tasks into an appropriate structure to accomplish the program purpose (structure charts).

Documentation is nothing more than recording the results of the design process so that it can be used to produce the code.

The steps in the data base group are design and create. (The data base we are talking about here is the test data base to be used to test the program.)

IV. Execution Of The Plan

The initial plan and schedule was followed with remarkable accuracy. The Project Standards Manual was late in completion, but the basic procedures had been verbally agreed upon by the team and documenting them seemed much less important than designing the data base.

A major deviation from the plan occurred very early when it was decided not to maintain the MRSS data structure but rather to redesign the file structure to eliminate redundancy. Surprisingly enough this change did not in any way impact the schedule.

A second major deviation from the plan was the relative infrequency of structured walk throughs which were held only just before wrapup of the analysis of a module and then only if the programmer asked for it. Structured walk throughs were not pushed by the lead analyst because the two programmers were consulting each other on a daily basis and a free interchange of problems and solutions was taking place without the more formal walk through procedure. Walk throughs were held when needed and were valuable in resolving some sticky problems but for the most part were not needed.

Another deviation from the plan was the discontinuance of interface charts. In the early stage of the project these charts were carefully maintained and appeared to be useful, but as the number of modules began to expand rapidly, it was found that the value of the charts diminished and the effort to maintain them increased so they were discontinued.

The most significant deviation from the initial plan came late in the program. Although DMS-170 was purchased early in the project it was not implemented until the project was in its last few months. Experimenting with Query rapidly led the team to the conclusion that the report generating portions of the system not only could be handled by Query, but that this part of the effort could be significantly reduced and system performance enhanced by use of Query.

One FORTRAN V problem was not resolved until after the first release of MRMS. The write end of record command did not work. This command was essential to one of the MRMS functions--a direct line teletype output of the schedule for downrange stations. The first version (released but not implemented) did not contain this broadcast capability. As soon as this command was working a second release activated the broadcast capability.

V. Results, Evaluation, Comments

The result of this effort are well summarized in the Verification Test Report which is included here in full.

PCF #Y109-[illegible]
KC34-[illegible]

VERIFICATION TEST REPORT

MECHANIZED RESOURCE MANAGEMENT SYSTEM

(MRMS)

16 October 1981

I. PURPOSE OF PROGRAM

MRMS is an interactive data base management system designed to maintain and update the MRMS resource/schedule data base. The program is divided into ten more or less independent command modules. Each command module recognizes and executes a certain logical subset of the MRMS command language. The following command modules make up MRMS:

1. JON Commands
2. Card Commands
3. Resource Commands
4. OD Commands
5. Number Commands
6. Operation Commands
7. Schedule Commands
8. Conflict/Commitment Commands
9. File Commands
10. Miscellaneous Commands

Modules 1 through 6 are data base update commands for maintaining the data base. Modules 7 and 8 produce specialized reports. Module 9 controls the handling of any print files created by the program. Module 10 contains various small commands used to control program execution parameters (e.g., default print files, schedule/history mode, etc.).

II. VERIFICATION

Verification was carried out in two parts. The first part consisted of providing Range Scheduling with a version of the program for their own use and verification. Some time was spent in training Scheduling personnel in the use of MRMS.

The second part of verification consisted of dividing the various command modules among three analysts for complete checkout and verification. Each module was checked out by an analyst who had done no programming in that particular module.

Checkout of the command modules consisted of using the MRMS User's Guide as a standard. Each data field defined by the user's guide was entered with the ADD command and modified with the CHANGE command, where applicable. Fields were tested from both the command line and from edit mode. Printouts of the data base were ordered before and after any data manipulation and were used to verify that the data base was correctly updated. Additionally, data dumps produced by Query-Update were compared to program generated printouts.

In checking out the OPERATION commands, the week of August 31 through September 6 was chosen to be entered by hand into the data base. This week consisted of 224 operations and provided a very thorough checkout of the scheduling portion of the program.

Two module commands, FILE and MISC, were not individually verified, but were used in all the other module chechouts.

III. RESULTS

In each command module all data manipulation commands were found to conform to the user's guide description. Printouts of the data base showed that the modified fields were in fact correctly updated. Query-Update dumps and printouts matched those produced by the program.

Comparison of the schedule for the week of August 31 to September 6 showed that the MRMS schedule compared very closely to the MRSS schedule. Any differences found consisted of either typing errors or neglect to change the default resource time spans obtained from an OD.

When comparing the conflict analysis reports, several major differences were found. These differences were expected, however, and were caused by three reasons.

1. Mismatching time spans between the two data bases. In entering so much data by hand, it was inevitable that some typos had occurred.

2. The NIB (Non-Interference Basis) flag eliminated many "false" conflicts in the data base.

3. Conflicts in MRSS were based on 15-minute intervals beginning between quarter hour divisions. In MRMS 15-minute intervals were used also, but began on the quarter hour divisions rather than between. This only effects "borderline" conflicts where there is resource usage in the same 15-minute block, but no actual overlap.

All discrepancies found in the conflicts printouts can be attributed to one or more of the above reasons.

No problems occurred when using the FILE or MISC command modules.

Range Scheduling reported the following problems from their use of the program:

1. No way to quickly delete all resources for an operation.

2. Command input line was rather short for convenience.

3. Scrub code validation did not allow numerics.

4. Remark mode inconvenient.

5. Some change in the regular and inter-range regular publish schedule report was needed.

The following changes were made in the program in response to Range Scheduling's remarks:

1. A delete all resources command (RESOURCE=*) was added.

2. The command input line was increased to three terminal screen lines (240 characters). This allows almost all parameters to be entered in one entry, using the edit mode only for major changes and errors.

3. Scrub code validation now allows alphanumerics.

4. Adding and changing remarks can now be done without the intervening step of using the Remark Edit Mode. Extensive remark editing can still use the Remark Edit Mode.

5. The requested changes were made in Publish Schedule format. These changes did not effect the downtime version of the Publish Schedule.

The above changes were made in the program, and verification was repeated for the portions of the program effected by the changes. It was found that the changes worked as specified.

Checking back to see how well our goals were met we found the following:

1. On-time completion. Almost. The 1-year slippage of the implementation date for the direct cost reimbursable/unit service change allowed some relaxation at the end of the effort. Our first release was 2 weeks late.

2. Provide all capabilities. Met--including the new ones generated by in-process interaction with the user.

3. Responsive to user's need and desires--met (see 2. above and the Verification Test Report).

4. Maintainable program. Only time can confirm this but this program has all of the proprieties defined in the literature as proper design for maintainability.

5. Efficient use of storage. Elimination of redundancy from the data base met this goal.

6. Code in a single style. Met--but the individual programmer's unique trademarks can be detected if you look hard enough.

The real payoff is the answer to the question of applicability of these techniques to other projects. Unfortunately, few firm conclusions can be drawn at this time. It is sure that the same team would handle any project of this type at least as well as they did this one. Other teams may have done as well on this project, some other teams would surely have failed. In any project the people are the most important ingredient. Good tools help but tools are valueless until properly used. What has been demonstrated by this project is that the tools used were good tools for this type of project. Their universal applicability is still much in doubt--but might be well worth tryinq.

Programmer productivity for this project was excentionally high. The total time spent on the project was 5087 man-hours, 33017 lines of code were produced. This yields a productivity value of 52 lines of code per day per programmer. A truly exceptional figure especially when it is remembered that those are both inexperienced programmers who learned not only FORTRAN V but also the DMS-170 language in the time charqed to this project.

MRMS PROJECT STATISTICS

Estimated Man-hours	6340
Man-hours Actual	5087
Percent Underrun	19.76

Program Statements Produces

A.	FORTRAN V	28090	
B.	CCL procedure Lines	580	
C.	Query Commands	3172	
D.	Conversion Programs (DB)	1125	
	Total	33017	(52 lines/Man-day)

Documentation (Permanent)	Pages
Project Standards Manual	31
User's Manual	42
Maintenance Manual	822
Verification Test Report	4
SDJ Report	24
	923

In-Process Documentation

- Structure Charts
- Interface Charts
- Status Reports

REFERENCES

1. Tom DeMarco, Structured Analysis and Systems Specifications, Prentice Hall.

2. Glenford Meyers, Composit/Structured Design, Van Nestrand Reinhold Company.

3. James Martin, Computer Data Base Organization, Prentice Hall.

4. Kernighan and Plauger, The Elements of Programming Style, McGraw-Hill.

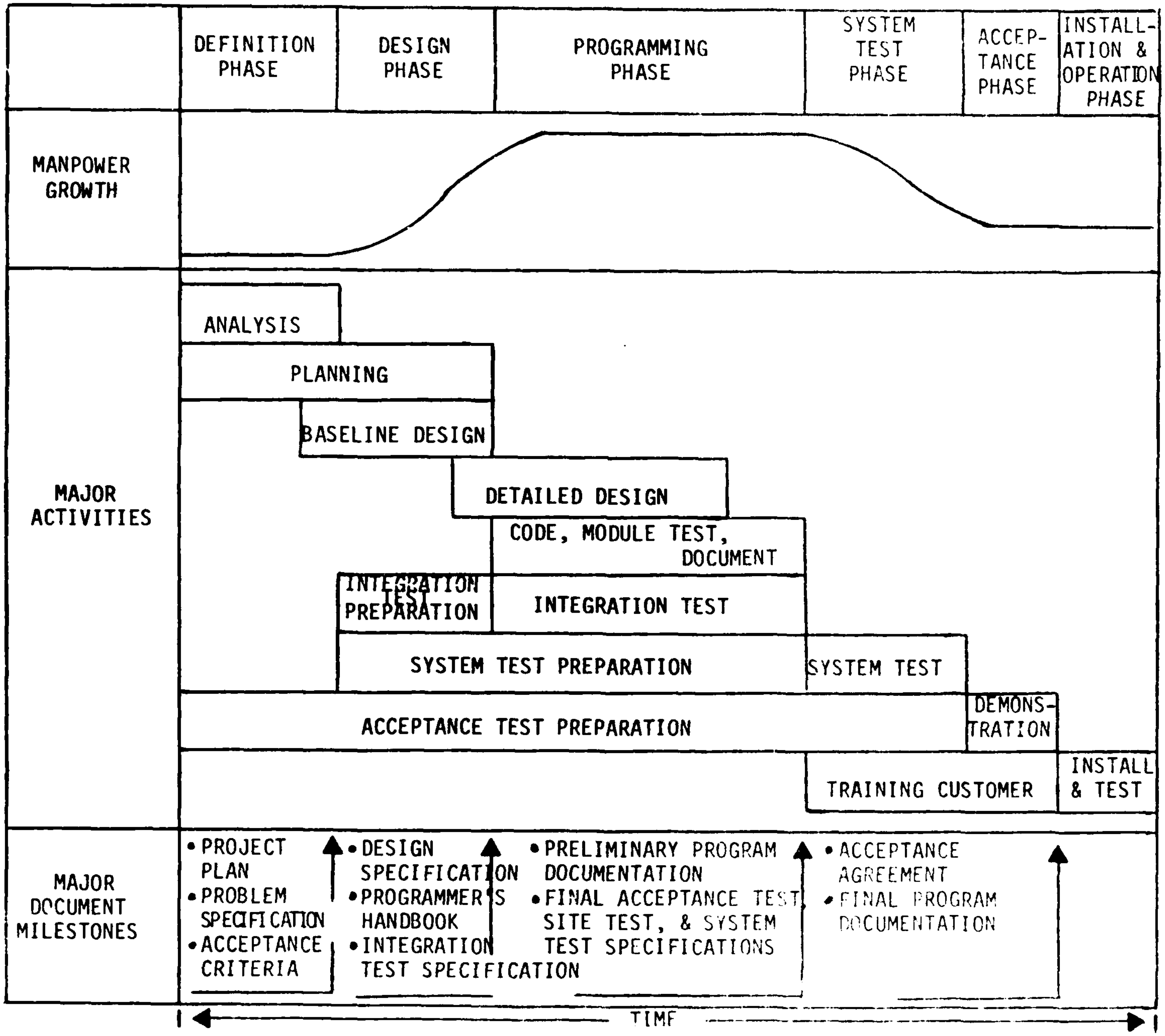
THE PROJECT - MECHANIZED RESOURCE MANAGEMENT SYSTEM (MRMS)

THE SUBJECT PROJECT WAS TO DESIGN AND IMPLEMENT AN INTERACTIVE SYSTEM TO:

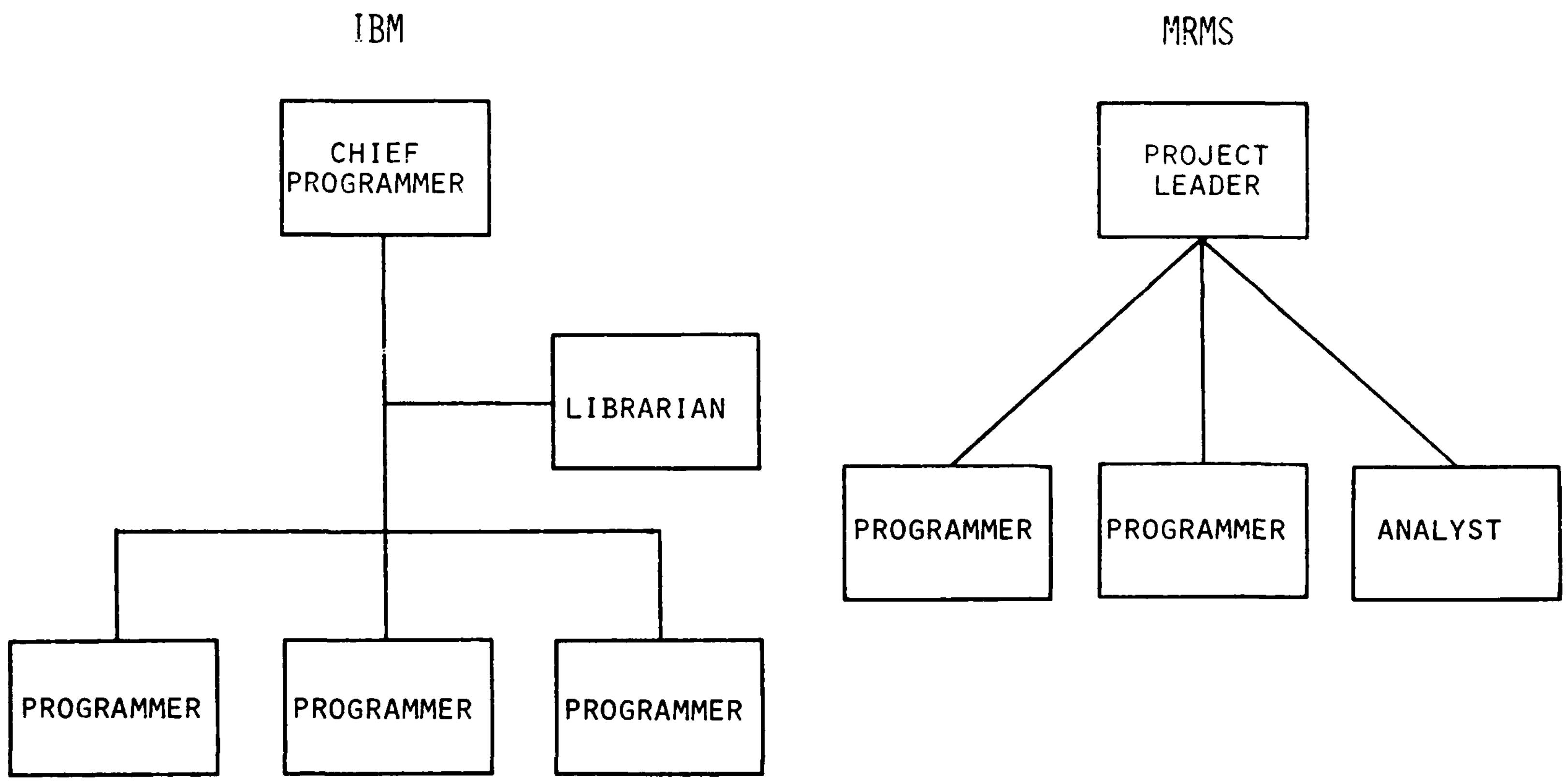
- SUPPORT ETR RANGE SCHEDULING OPERATIONS
- PROVIDE ACCURATE AND ACCOUNTABLE RESOURCE UTILIZATION HOURS IN SUPPORT OF DIRECT COST REIMBURSEMENT USING A UNIT SERVICE CHARGE CONCEPT

ETR WAS COMMITTED TO IMPLEMENT REIMBURSEMENT UNDER UNIT SERVICE CHARGE AT THE START OF FY82.

PROGRAM DEVELOPMENT CYCLE



DEVELOPMENT TEAM



PROJECT GOALS

- On time completion
- Provide all required capabilities
- Responsive to Users needs and desires
- Maintainable program
- Efficient use of storage
- Code in a single consistent style

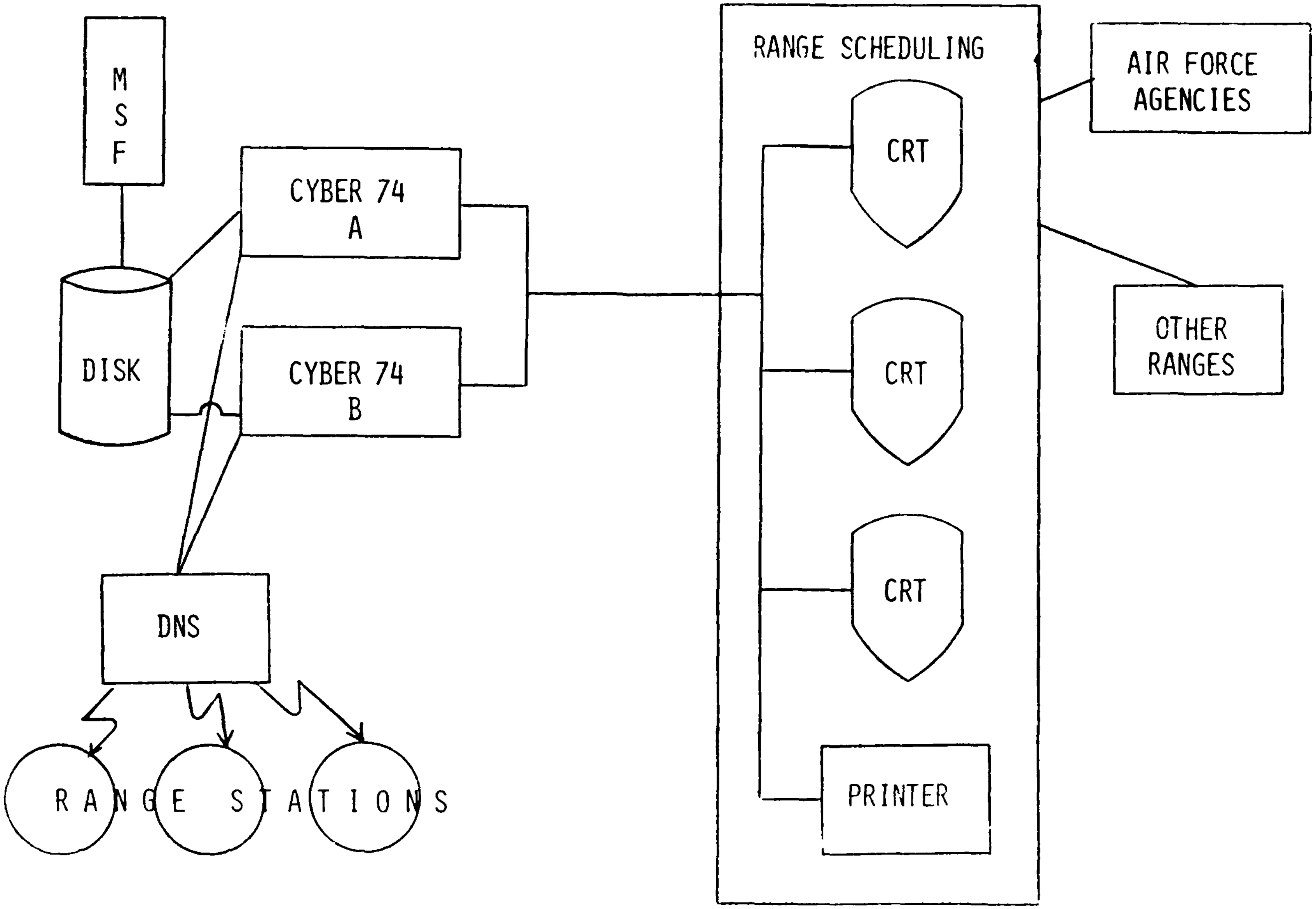
TOOLS

- FORTRAN V
- STRUCTURED PROGRAMMING
- STRUCTURED COMPOSITE DESIGN
- STRUCTURED CHARTS
- INTERFACE CHARTS
- DMS 170
- ALGORITHM STATEMENT INSTEAD OF FLOW CHARTS
- STRUCTURED WALK THROUGHS

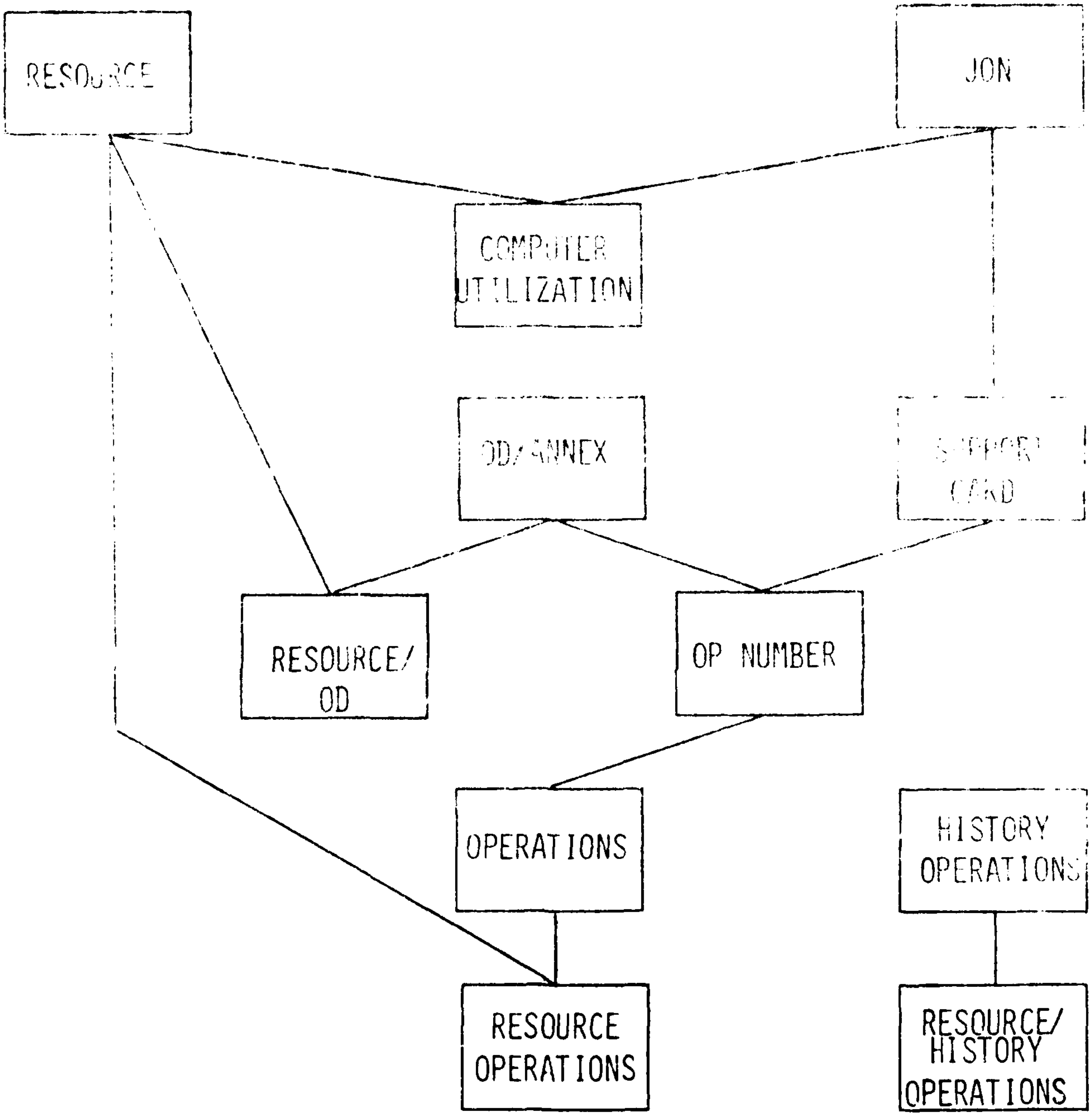
SEQUENCE OF TASKS

- WRITE PROGRAMMING STANDARDS MANUAL
- DESIGN DATA BASE
- DESIGN MAN-MACHINE INTERFACE
- WRITE USERS MANUAL
- DEFINE MAJOR PROGRAM SECTIONS
- DESIGN UTILITY ROUTINES
- DOCUMENT UTILITY ROUTINES
- CODE UTILITY ROUTINES
- DESIGN TEST DATA BASE
- CREATE TEST DATA BASE
- DETAIL STRUCTURE DESIGN OF MAJOR SECTIONS
- DOCUMENT MAJOR SECTIONS
- CODE MAJOR SECTIONS
- ACCETPANCE TESTING
- USER TRAINING
- PROGRAM CERTIFICATION

FUNCTIONAL HARDWARE AND INTERFACE



MRMS DATA BASE OVERVIEW



TEAM ACTIONS

- DESIGN REVIEW
- STRUCTURED WALK-THROUGH
- DOCUMENTATION
- CODE
- TEST
- ACCEPTANCE
- USER TRAINING
- CERTIFICATION

MRMS PROJECT STATISTICS

PROGRAM STATEMENTS PRODUCED

A.	FORTRAN V		28090
B.	CCL PROCEDURE LINES		580
C.	QUERY COMMANDS		3172
D.	CONVERSION PROGRAMS (DB)		1125
		TOTAL	33017 (52 LINES/MAN-D

DOCUMENTATION

		PAGES
PROJECT STANDARDS MANUAL		31
USER'S MANUAL		42
MAINTENANCE MANUAL		822
VERIFICATION TEST REPORT		4
SDJ REPORT		24
	TOTAL	923

MANPOWER

PROJECTED	6340
ACTUAL	5087

SCHEDULE

PROJECTED COMPLETION	1 OCTOBER 1981
ACTUAL COMPLETION	16 OCTOBER 1981

GOAL ACCOMPLISHMENT

- On time completion
- Provide all required capabilities
- Responsive to Users needs and desires
- Maintainable program
- Efficient use of storage
- Code in a single consistent style

SOFTWARE ACQUISITION
WITHIN A
SYSTEM ACQUISITION

by

JOHN GREENWALD
HEAD, DATA PROCESSING SYSTEMS
CONTROL SYSTEMS DEVELOPMENT DIVISION
RANGE DEVELOPMENT DEPARTMENT
RANGE DIRECTORATE
PACIFIC MISSILE TEST CENTER

presented at

SOFTWARE ENGINEERING SEMINAR
DATA REDUCTION AND COMPUTING GROUP
RANGE COMMANDERS COUNCIL
2 FEBRUARY 1981
NELLIS AIR FORCE BASE, NEVADA

This document has been prepared for information purposes only. The information contained herein does not necessarily representlу the official views, position, and policy of the Commander, PACMISTESTCEN and the U.S. Navy. The Commander, PACMISTESTCEN and the U.S. Navy assumes no responsibility for action taken on the basis of this document.

ABSTRACT

This paper describes software acquisition within a system acquisition from the point of view of the purchaser. The content is scoped for acquisition of systems which have functionally embedded computer systems such as tracking, target control, process control, information systems, telemetry processing, and data communications.

INFORMATION SOURCE STATEMENT

The material contained in this paper represents my current concept of the process of software acquisition within a system acquisition from the point of view of the purchaser. The material represents my conclusion based on years of experience as a computer and software systems engineer for system acquisition.

John Greenwald 24 Jun 1982

John Greenwald

TABLE OF CONTENTS

TABLE OF CONTENTS

1.0 INTRODUCTION

This paper describes software acquisition within a system acquisition from the point of view of the purchaser. The material contained within this paper is scoped for acquisition of systems such as tracking, target control, process control, telemetry real-time processing, command and control, and realtime information translator being developed under a cost type contract where delivery is approximately 1 to 3 years after contract award. The term software in this paper includes firmware.

Section 2.0 of this paper describes the system acquisition process for a typical system containing software with emphasis on the software items. Section 3.0 describes the contents of documents which define system requirements and define what information should be submitted by prospective contractors for proposal submittal and by the contractor during the life of the contract. Section 3.0 also emphasizes the software items.

2.0 SYSTEM ACQUISITION PROCESS

The system acquisition process consist of a number of phases and evaluation periods. Figure 1 shows the major phases and evaluation for a typical system containing software. The time required for each phase and evaluation is a function of the system being acquired and the contract conditions. Also, figure 1 shows phases and evaluations which may or may not be effectively overlapped.

2.1 System Recuirements Phase

The system requirements phase is the initial phase. This is the phase when the need for the system is established. The requirements may either be functional statements or describe a specific type of system. An example is "A tracking system with the characteristics that follow is required:" or "A radar with the characteristics that follow is required:" Software in itself is not generally stated as a system requirement.

2.2 Procurement Documents Preparation Phase

A number of documents are required for a technical effective, cost efficiency, and manageable system acquisition. The procurement documents define the technical requirements for the system and define the rules by which technical progress is reported, design reviews are performed, testing is performed, quality is established, end items are accepted, etc. After all procurement documents are completed, prospective contractors are requested to submit a proposal which describes how they would meet the requirements.

The procurement documents which involve software items of a system are described in section 3 and are:

a. Proposal Preparation Requirements

b. Deliverable End Items Descriptions

c. Technical Specifications

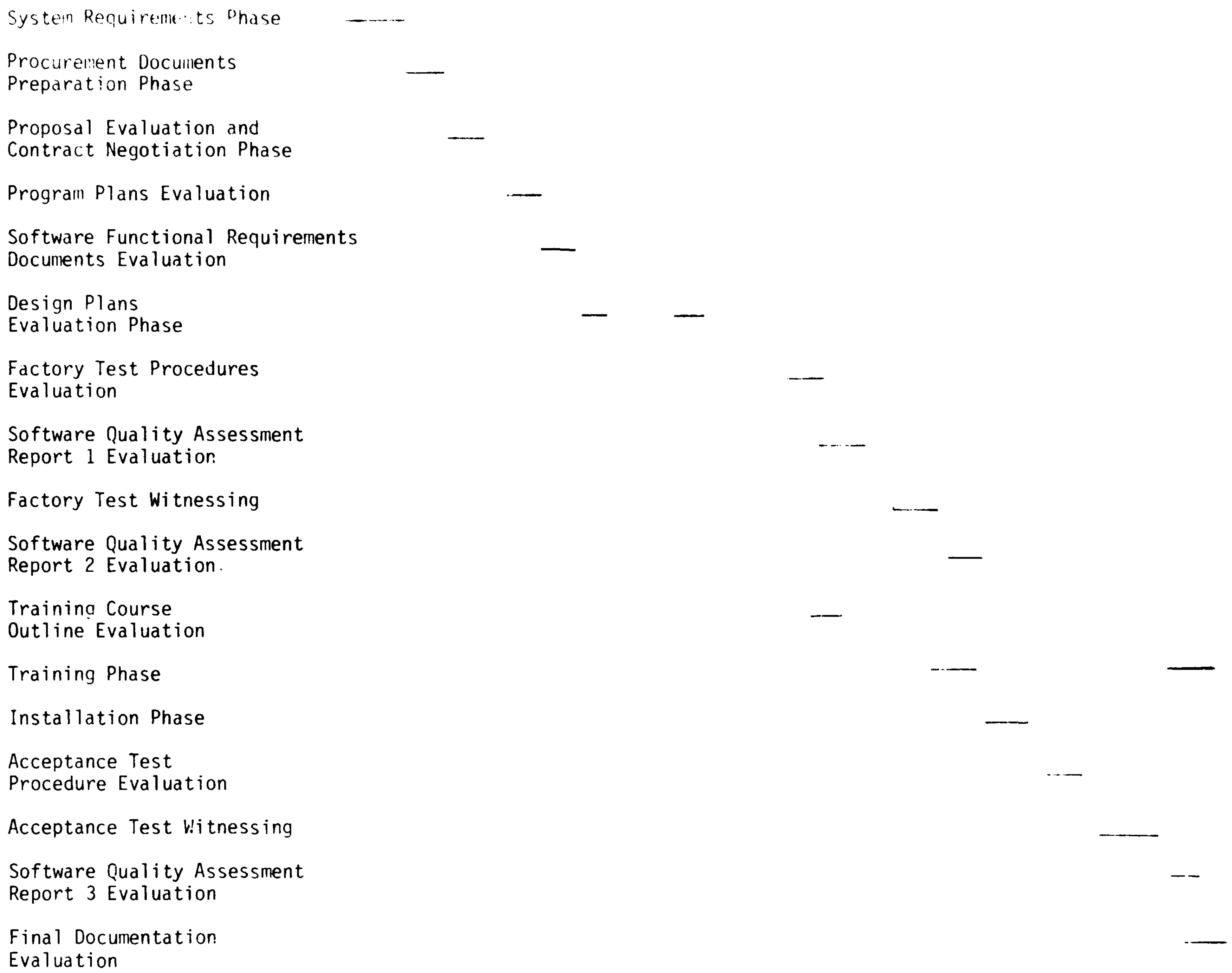


Figure 1 - SYSTEM ACQUISITION [illegible]

d. Training Specifications

e. Deliverable Data Items Descriptions

Data items are documents by which the contractor reports progress, describes his management approach, describes the design, reports costs, defines test procedures, describes the system as built, etc. to the purchaser during the life of the contract. Procurement data items which involve software items are:

a. Progress Reports

b. Software Implementation Program Plan

c. Software Quality Assurance Program Plan

d. Software Functional Requirements Document

e. Software Human Interface Design Document

f. Software Functional Design Document

g. Software Operating Procedure Document

h. Software Implementation Design Document

i. Software Implemented Design Document

j. Software Quality Assessment Report

k. Software Factory and Acceptance Test Procedures

l. Training Course Outline Document

2.3 Proposal Evaluation and Contract Negotiation Phase

During this phase the proposals submitted by each contractors are evaluated for compliance to the requirements and the proposal preparation requirements, and a contract is negotiated and awarded.

Generally, for a system procurement, technical requirements are stated for the system rather than as requirements for software end items. With this approach, the evaluation must be concerned with what is proposed to be implemented in software, the proposed structure of the software, the selection of the programming languages when languages were not specified, the estimate of the size of the software, the estimate of the processor requirements to execute the software, the capabilities of the entire proposed computer system for providing the capabilities necessary for the software to perform its functions, and the proposed software testing methods.

The proposal preparation requirements usually require a management proposal which describes contractor organization, allocation of resources for producing the system, schedules for achievement milestones, schedules for deliverable items, and procedures for achieving end items such as the procedures for implementing software and assuring software quality. The evaluation must be concerned with the reasonableness of the estimated manhours for software development, effectiveness of the proposed software organization, effectiveness of the proposed software manloading for each element of software development, reporting structure of the software organization within the entire organization, and visibility of the software commensurate with its criticalness and magnitude.

For the training requirements, the evaluation must be concerned that the training courses are complete and of the proper technical level and required type.

2.4 Program Plans Evaluation

Shortly after contract award, the contractor is generally required to submit program plans for producing the system. The primary program plans which involve software are the Software Implementation Program Plan and the Software Quality Assurance Program Plan.

For the Software Implementation Program Plan, the evaluation is concerned with effective utilization of resources and the procedures to be used to produce each software end items, associated data items, and meet the achievement milestones. This plan should be kept current and specifically revised following submittals of the functional design for each software item.

For the Software Quality Assurance Program Plan, the evaluation is concerned with the procedures to be used to ensure that the software end items meet the requirements, ensure that associated documentation describes the software items as designed and built, and estimate the expected value of the mean time between error occurrence and its variance.

The evaluation of both plans is concerned with the relationship and interface between personnel implementing the software and software quality assurance personnel.

2.5 Software Functional Requirements Document Evaluation

The evaluation of the Software Functional Requirements Document is concerned with:

a. Completeness and correctness of the Software Functional Requirements Document.

b. Feasibility of implementing the functional requirements in software.

c. Effectiveness of implementing the functional requirements in software.

d. Compliance of the software functional requirement to the contract system requirements.

2.6 Design Plans Evaluation Phase

The Design Plans Evaluation Phase should consist of a sequence of design review steps where the design level content of the design plans increases with each design review until the design level necessary for design approval is reached. There must be sufficient time between each design review for the contractor to respond to the design review evaluation reports and to increase the design level content of the design plans. Each software end item should have a separate set of design documents. The documents required for a two step design review sequence are as follows:

Design Review I Documents

a. Revised Software Implementation Program Plan

b. Revised Software Functional Requirements Document

c. Software Human Interface Design Document

d. Software Functional Design Document

Design Review II Documents

a. Revised Software Implementation Program Plan

b. Revised Software Functional Requirements Document

c. Revised Software Human Interface Design Document

d. Revised Software Functional Design Document

e. Software Operating Procedures Document

f. Software Implementation Design Document

The revised documents shown are necessary for completeness of the design review and for revising the Software Implementation Program Plan to be consistent with the current design.

The Software Human Interface Design is evaluated for its effectiveness and operator error probability.

The Software Functional Design is evaluated for:

a. Completeness and correctness of the translation of the software functional requirements.

b. Feasibility and applicability of the functional breakdown of the software item.

The Software Operating Procedure is evaluated for completeness, approach, understandability, and ease of operation of the software item.

The Software Implementation Design is evaluated for:

a. Completeness and correctness of the translation of the software functional design.

b. Structure, understandability, and modifiability of the software item.

2.7 Factory Test Procedures Evaluation

The primary factory test procedures involving software are the software test procedures. The evaluation is concerned with:

a. The design of the test to demonstrate compliance to requirements.

b. The data collected during the test for analysis of data products and human interface.

c. The test demonstrating the estimated value of the expected time to error occurrence for critical software items.

d. The effects of using simulated data sources and interface devices.

2.8 Software Quality Assessment Report 1 Evaluation

The Software Quality Assessment Report 1 should be submitted before start of factory test. The report should contain quantitative measurements and functional analysis for each critical software end items. The evaluation is concerned with the quality assessment of the software items for compliance with the functional requirements and for the estimate of the expected execution time between error occurrences.

2.9 Factory Test Witnessing

Factory Test Witnessing is the process of actually observing the system perform under a test environment. This includes monitoring elements of the system and anlysis of data produced by the system. The tests should be those defined in the Factory Test Procedures Document previously submitted and approved. All software end items should be tested to demonstrate their compliance to requirements. For critical software end items, sufficient data should be collected to establish an estimated value for the expected time between error occurrences.

2.10 Software Quality Assessment Report 2 Evaluation

Software Quality Assessment Report 2 should be submitted after completion of the factory test and should be a revision of report 1 based of factory test results. The evaluation is concerned with the same items as the evaluation for assessment report 1.

2.11 Training Course Outline Evaluation

Of the training courses, generally, three courses are associated with software. The three courses are system introduction, system operation, and software. The evaluation is concerned with:

a. Technical level of each course.

b. Completeness of each course.

c. Applicability of the material to be covered in each course.

2.12 Training Phase

This is the phase where training occurs according to the submitted and approved training course outline.

2.13 Installation Phase

This is the phase where the system is installed at the required location.

2.14 Acceptance Test Procedures Evaluation

The evaluation of Acceptance Test Procedures is similar to the evaluation of Factory Test Procedures with greater concern for the validity of the tests. The acceptance test procedures are expected to produce more meaningful results than the factory test procedures. Therefore, the procedures used for the factory test are expected to be revised as necessary to improve the test.

2.15 Acceptance Test Witnessing

Acceptance Test Witnessing is similar to Factory Test Witnessing. For the acceptance test, the test environment should be real and the system should be required to be compliant to all requirements before the test is completed.

2.16 Software Quality Assessment Report 3 Evaluation

Software Quality Assessment Report 3 should be submitted after completion of the accepted test and should be a final assessment report based on the acceptance test and all previous test. The evaluation is concerned with the same items as the evaluation for assessment reports 1 and 2.

2.17 Final Documentation Evaluation

Software final documentation should include a Software Implemented Design Document and the Operating Procedure Document for each software end item. The evaluation is concerned with the completeness, correctness, and usefulness of the documents.

3.0 PROCUREMENT DOCUMENTS DESCRIPTIONS

This section describes the content of documents which involve software which are part of a system acquisition procurement document package. These documents describe system requirements and what information should be submitted by prospective contractors for proposal submittal and by the contractor during the life of the contract.

3.1 Proposal Preparation Requirements

The Proposal Preparation Requirements defines what information should be contained in a proposal. The Proposal Preparation Requirements should require for each software item the following:

a. A description of the functional requirements for the software item including those imposed by the system implementation approach and the technical specification written in non-programming terminology.

b. A top level functional design.

c. A description of the structure proposed for implementing the software item.

d. Identification of proposed programming language.

e. An estimate of the computer resources required by the software item including processor time, memory size, input bandwidth, output bandwidth, secondary memory, and peripheral devices such as tape transport terminals, etc.

The Proposal Preparation Requirements shoulc require descriptions of the contractor's software organization for producing the software items, allocation of resources, and schedules for achievement milestones and deliverable items.

3.2 Deliverable End Items Description

Deliverable End Items are the physical subsystem which make up the physical system, software items, training, repair parts, etc. Generally, the end items need to be defined before the final structure and content of the other procurement documents can be defined. The basic reason for this is that the other procurement documents must be structured to require the end items. All software items for which requirements are defined in the technical specification should be titled and defined as deliverable end items.

Often the approach proposed by the contractor for a system contains a computer in an element of the system where no specific software requirements were stated in the technical specification. The non-specified software items necessary because of the contractors approach should have the same data items and delivery requirements as for specified titled software end items. There are two apparent approaches for handling this problem. An end item called "Non-Titled'Software" can be defined as an end item or an assumption can be made that all subsystem contain software. For the latter, a software end item would be defined for each subsystem.

3.3 Technical Specification

The technical specification defines the requirements for the physical subsystems and software items which make up the subsystem. Each identifiable software items should be titled and its requirements defined. Often for systems such as tracking, process control, etc. performance requirements are stated for the system rather than as requirement for a software item. The basic reason for this is to allow the contractor to propose the most effective and efficient method of meeting the system requirements. When this approach is used, software features which are considered necessary for system testing, system monitoring, performance evaluation, and modification such as: recording input data, recording output data, recording operator interactions, system status, system error messages, etc. but are not necessary for system performance must be specifically required.

Where specific programming languages are required, they must be specified. Caution must be used in specifying a language requirement such that it is clear if the language requirement applies to the software of devices such as communication controllers, display controllers, disk controller, etc.

Basically, there are four approaches which can be used in specifying operating system requirements which are:

a. No operating system for system operation and language translation support is provided by another computer system.

b. No operating system for normal system operation, but one used for software support requirements.

c. One operating system used for normal system operation and another used for software support requirement.

d. One operating system to be used for all software items except processor diagnostics.

All identifiable support software requirements should be specified. Where multiple languages are allowed, a statement like the following is needed "A language translator is required for each software item of the system." Generally, a diagnostic should be required for each component of the computer and each device interfacing the computer.

To cover software items which cannot be identified until the system implementation approach is known, used a statement like the following "All software items necessary to use, load, store, and modify the software required to meet the requirements specified herein shall themselves be software end items."

3.4 Training Specification

The training requirements must be scoped for operating personnel for operation and maintenance and for engineering personnel for modification of the hardware and software of the system.

The Training Specification should contain requirements for a system introduction course, system operation course, software course, and other appropriate courses. Generally, personnel who will attend the software course should first attend the system introduction course and the system operations course. Therefore, the three courses which software personnel are expected to attend should be required to be scheduled not to overlap in time.

The system introduction course should be scoped as a prerequisite for all other courses. The system operations course should be a completed in depth course on procedures to operate the system and how to effectively utilize the system. The software course should be required to cover the software functional requirements, software functional design, software top level implementation [illegible] and the software quality assessment procedures and tools for each software end item.

The specification should indicate the types and levels of personnel to receive each type of training, and also clearly indicate if training is being required on the programming languages used to implement the system.

3.5 Deliverable Data Items Descriptions

Deliverable Data Items are documents by which the contractor reports progress, report his management approach, describes the design, defines test procedures, describes the system as built, etc. to the purchaser during the life of the contract. The following paragraphs describe the content of data items which should be required for all software items.

3.5.1 Progress Reports

Progress Reports should be submitted at a defined time interval. The software area of the progress report should report progress for each software end item separately. Progress should be reported against the achievement milestones defined in the Software Implementation Program Plan which identifies the elements that comprise each software end item and associated data items. For each software item, a report should contain:

a. Record of achievement milestones reached.

b. Record of manload versus time and milestones.

c. Record of cumulative manhours used.

d. An estimate of manhours to reach next milestone.

e. An estimated of manhours to complete software item.

f. Identification of problems.

3.5.2 Software Implementation Program Plan

The Software Implementation Program Plan should be submitted shortly after contract award and should describe:

a. The software engineering organization for producing the required software end items and associated data items.

b. The interface between the personnel implementing the software items and the software quality assurance personnel.

c. Meaningful achievement milestone for each software item and associated data items for which progress should be reported against in the progress report.

d. The procedures to be used to produce each software end item and associated data items and to meet the achievement milestones on schedules.

e. Manloading versus time in tabular and graphical form with the achievement milestones indicated.

The plan should be revised following the initial and each revision to a Software Functional Design. The revised plan should show work starting and work completed for each process which is scheduled by the operating system, or interrupt driven, or invoked by the root process of the software item as an achievement milestone.

3.5.3 Software Quality Assurance Program Plan

The Software Quality Assurance Program Plan should be submitted shortly after contract award and should describe the procedures for:

a. Assuring that the Software Functional Requirements are correctly and completely difined.

b. Assuring that the Software Functional Design is a correct and complete translation of the Software Functional Requirements.

c. Assuring that the Software Implementation Design is a correct and complete translation of the Software Functional Design.

d. Assuring that the Software Implemented Design describes the software as built.

e. Detecting, recording, analyzing, and correcting software deficiencies.

f. Assuring that the code form of the software item is correct and a complete translation of the Software Implementation Design and is compliant with requirements.

g. Collecting and reporting the following information for errors:

(1) Execution time between error occurrence.

(2) Type and severity of error.

(3) Corrective action taken.

h. Estimating the mean time between error occurrence and the variance in the estimate.

The plan should define the interface between the software quality assurance personnel and the personnel implementing the software.

3.5.4 Software Functional Requirements Document

A Functional Requirements Document should be required for each software end item and should not be written in programming terminology. The Functional Requirements Document should describe what the software end item will accomplish, when the accomplishments will occur, and under what constraints and conditions the accomplishments will be achieved. The Functional Requirements Document should contain the following information:

a. The data products which must be produced.

b. The relationship and dependencies between data products.

c. The requirements for each mode of operations.

d. List of devices, physical subsystem, etc. which must be controlled.

e. The interactive and setup control which must be provided.

f. The human readable information which must be provided.

h. The relationship and dependence between data input, data output, input control information, and output control information.

i. The purpose of each interfacing device.

j. The relationship and dependence between interfacing devices.

k. The data inputs which must be accepted from each interfacing devices and when the inputs must be accepted.

l. The purpose of each data input or output from each interfacing device.

m. What must be done with each data input.

n. What must be done to produce each data output.

o. The data outputs which must be provided to each interfacing device and when the output must be provided.

3.5.5 Software Human Interface Design Document

A software Human Interface Design Document should be required for each software end item which has an on-line interface via a device to a human being. The Software Human Interface Design Document should describe how human beings interface with the software end item and should contain the following information:

a. Description of the procedures an operator will perform to set up and interact with both control and product displays and generate hard-copy records of displays.

b. Structure of interactive and setup control input and output information such as: alphanumeric display images, graphic display images, keyboard commands, card formats and hardcopy outputs.

3.5.6 Software Functional Design Document

A Software Functional Design Document should be required for each software end item. The Software Functional Design Document should describe how the requirements for the software end item will be accomplished and what software structure will be used for the software end item. The Functional Design Document should contain the following information:

a. Functional diagrams which identify and show the relationships between processes and data structure for all processes which are scheduled by the operating system, interrupt driven, or invoked by the root process of the software item.

b. Description of each identified process and data structure.

c. Description of the control structures used to schedule processes to run, suspend and terminate.

d. Description of the protection mechanism between processes, data structures, and control structures.

e. Identification of the data structures requiring data access synchronization.

f. Interface description for each hardware device as seen from the software item.

g. State-transition diagrams for each multiple state device and entities such as : communication path, communication device, tracked object and other entities.

h. Identification of the programming language for each process and data structure.

3.5.7 Software Operating Procedure Document

An Operating Procedure Document should be required for each software end item. The Operating Procedure Document should evolve from the Human Interface Design and the Functional Design. The Operating Procedure Document should contain a completed description of how to effectively utilize the software item, how to obtain the data products of the software item, the formats and contents of operator interface items such as: alphanumeric display images, graphic display images, keyboard command structure, card formats, and hardcopy output formats.

3.5.8 Software Implementation Design Document

A Software Implementation Design Document should be required for each software end item. The Software Implementation Document should evolve from the function design and should describe how the functional design is to be implemented, what structure is to be used at each design level, and how each design level is to be implemented. The Implementation Design should contain the following information:

a. Diagrams which show the relationship and dependence between the processes, data structures, and control structures at each design level.

b. Diagrams which show and narrative which describes how each process, data structures, and control structure are implemented.

c. Hierarchial tree diagrams of the processes which comprise each software item

d. Interrupt structure of the software item.

e. Interprocess communication method.

f. How each process is scheduled or invoked.

g. Process synchronization methods and data access synchronization methods.

h. Format and content description of all data and control information input to and output from the software item.

i. Detailed format and description of each data and control structure contained within the software item at each design level containing title of the structure and each sub-structure, type of structure, and content description.

i. A description for each process contained within the software item at each design level which consist of: title, what the process must accomplish, when the accomplishment must occur, re-entrant or non-reentrant, mathematical model, definition of all inputs and outputs, linkage used when invoking other processes, and diagrams which shows the flow and logic of the process.

3.5.9 Software Implemented Design Document

The Software Implemented Design Document contains the same information as the Software Implementation Design Document. The difference being that the Implemented Design describes the software item as built, whereas the Implementation Design describes how the software is intended to be built.

3.5.10 Software Quality Assessment Report

A Software Quality Assessment Report should be required for each software item which is critical to the functioning of the system. The report should contain the following information:

a. Records of errors occurring during testing which contain:

(1) Execution time between error-occurrences.

(2) Type and severity of error.

(3) Corrective action taken.

b. Estimate of the expected time between error occurrences and the expected variance of the estimate.

3.5.11 Software Factory and Acceptance Test Procedures

A software test procedure should be required for each software end item. Critical software end items should be tested for sufficient time and under relistic conditions to substantiate an estimated value of the expected time to error occurrence. The test procedure for each software item should:

a. Be designed to demonstrate that the software item meets functional requirements.

b. Require data collection of and provide for analysis of data products and human interface data.

c. Prohibit modification to the software item during testing.

d. Require the test to start with the source language form of the software item and produce an executable form. This is necessary to ensure that the tested and delivered executable form is for the same version of the software item as the source form.

For the factory test procedure, the use of simulated data sources and data interface devices are acceptable where the devices are not realistically available. However, during acceptance testing, simulated devices cannot be used for the final integration test.

3.5.12 Training Course Outline Document

The Training Course Outline Document should provide sufficient description for evaluation and approval of proposed contents of each training course. The content of each Training Course Outline should be:

a. Course title.

b. Course objectives.

c. Course scope and level.

d. Course prerequisites.

e. Course content breakdown by topic and time per topic.

f. Course length in hours and number of sessions.

g. List of course publications such as: textbooks, documents, technical literature, etc.

FLIGHT TEST ORIENTED PRECOMPILER SYSTEM (FLTOPS)

by

Thomas R. Berard

Air Force Flight Test Center
6520th Test Group/ENCS
Edwards AFB, California 93523

ABSTRACT

FLTOPS is a general purpose software development tool designed to customize generalized FORTRAN software packages to a specific application. It features a readable hierarchical Input File to specify the desired options and a FLTOPS Configuration File which incorporates a specialized high-level language and embedded FORTRAN code to provide the necessary generalization.

I) INTRODUCTION

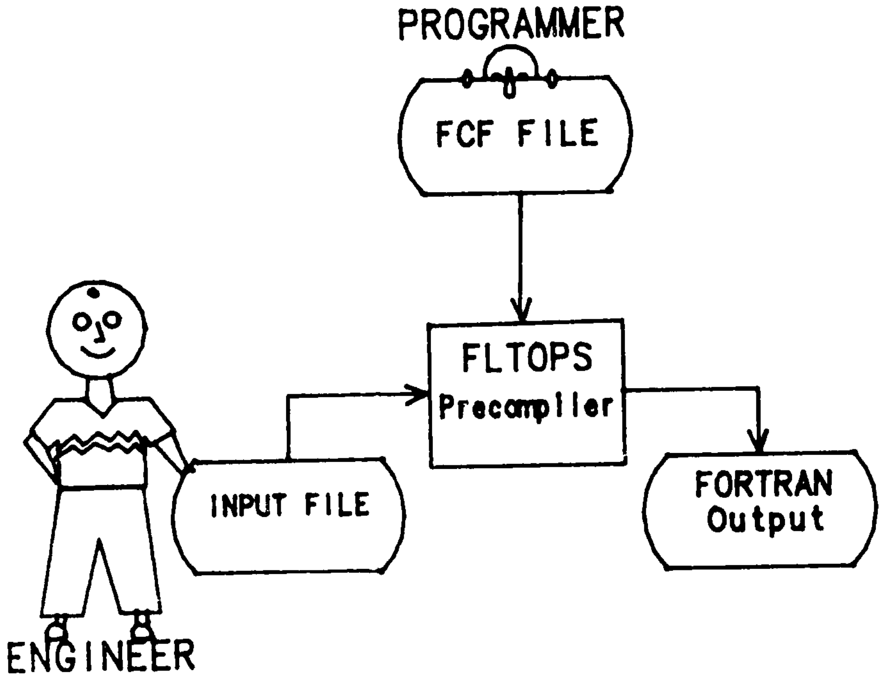
The Flight Test Oriented Precompiler System (FLTOPS) is a software tool used to assist in the rapid development and continued maintenance of customized software. It was developed under contract for the Air Force Flight Test Center (AFFTC), Edwards AFB, CA, by the Man-Computer Systems Division of Science Applications, Inc. (SAI), Englewood, CO.

The FLTOPS system was developed in response to a problem which exists in many application software domains, particularly in connection with flight test analysis activities at AFFTC. The development of specialized software for analysis of a new flight test program is an expensive and time-consuming operation. This software has many similarities to the software developed for previous flight test programs, and yet the differences are sufficient to prevent the use of a single, generalized software package. What is needed is a software development tool capable of rapidly producing specialized versions of a software package. The resulting software must be efficient and reliable, and must be produced, as automatically as possible, in response to the flight test engineer's statement of analytic requirements.

The goal is to have a tool which accepts information at the engineer's level (engineering specifications) and then automatically generates the appropriate software. Obviously, the system must include, in some form, considerable information about the mapping of engineering specifications into computer programs. It is desirable, though, that this information not be built into the precompiler itself, since that would unnecessarily restrict the range of software products for which the tool is useful.

To meet this goal, FLTOPS was designed to have two input files. (1) A hierarchical Input File contains the engineering specifications and drives the customization. (2) A generic "FLTOPS Configuration File" (FCF) defines the mapping from engineering specifications into computer programs. Essentially the FCF contains segments of the analytic software package and special high-level language commands which allow customization.

With this combination it is a relatively easy task for a programmer to take existing FORTRAN software packages and break them up into logical sections which can easily be included or excluded depending on the user's input. The size of arrays or the values of constants can also be left to the user or calculated depending on some user input, like the number of options chosen.


PROGRAMMER
FCF FILE
FLTOPS
Precompiler
INPUT FILE
FORTRAN
Output
ENGINEER

II) INPUTS

There are two inputs to the FLTOPS precompiler, the hierarchical Input File and the FLTOPS Configuration File (FCF). The first input, the Input File, is by far the most visible of the two.

The Input File provides the ultimate customization control and serves as the interface with the engineer. It has a hierarchical structure similar to an outline. This structure provides an easy to read document which describes all the options used to create the FORTRAN source code. Since it can be written in engineering terms, it may be used by project engineers to brief their superiors or to file away for historical reference. This outline-like format also provides a means of establishing several levels with options desired at each level. As an example, suppose various options are available to make corrections to measured aircraft angle of attack. The Input File then might contain the following specifications.

```
ANGLE_OF_ATTACK CALCULATIONS
   CORRECTIONS
      DYNAMIC_LAG
          .
          .
          .
      BOOM_BENDING
          .
          .
          .
      PITCH_RATE
          .
          .
          .
```

It is clear from the structure that dynamic lag, boom bending and pitch rate are corrections made in the angle of attack calculations. Under DYNAMIC_LAG, BOOM_BENDING and PITCH_RATE, additional specifications might be made about exactly how the calculations are to be made (e.g., specification of options and constants to be used).

Another application may not desire any corrections be made to the angle of attack. In that case the same portion of the Input File would then contain the following:

```
ANGLE_OF_ATTACK CALCULATIONS
   CORRECTIONS NONE
```

FORTRAN code from this example would not contain all of the complicated Angle of Attack corrections, thereby eliminating the unnecessary "baggage" throughout the life of the project.

The Input File can also be used to describe the defaults, functions, analytical equations and/or actual FORTRAN code to be used in the program.

```
ATMOSPHERIC CALCULATIONS
   USE MIL STD 210-A TROPICAL ATM WITH 1962 STD PRESSURE
   CONSTANTS
      RATIO OF SYSTEM LAG TO STATIC LAG = 1.02
      STATIC LAG PARAMETER = SLAMSL
   TEMPERATURE PROBE CORRECTION
      (TT/TA) = F(.2M**2)
         CURVE NUMBER = 5010
   OUTPUT FORMATS TO BE USED IN PLACE OF EXISTING ONES
      6501 FORMAT(1H ,10X,3A15)
      6502 FORMAT(1H ,10X,3F15.3)
           .
           .
           .
```

It became obvious during the testing phase of the functional design that the Input File could be very large, complex and be significantly different from FCF to FCF. Hence, the evaluation team decided that an interactive aid to guide the engineer in the building of the Input File would be beneficial to all concerned. Such a tool, the Interactive Input Processor (IIP), is currently being implemented at AFFTC. It receives an input which parallels the FCF and guides the engineer through the hierarchy of possible options. Though this is not a necessary step to use FLTOPS, it may be extremely helpful for large complex systems with many levels of options.

The Input File structure and allowable contents are dependent on a corresponding FCF. The FCF is essentially a program which reads the engineer's Input File and generates the desired FORTRAN source code. To do this the FCF is divided into two distinct, yet intermixed parts; (1) a new, high-level language to control the reading of the input and creation of the FORTRAN source code and (2) blocks of FORTRAN source code to be included or excluded depending on the user's inputs.

This new, high-level, FCF language is similar in form and design philosophy to many other modern languages such as PL/1 and ALGOL. It is block structured, stream oriented and has conditional and looping statements to allow diversion from sequential processing. Programs written in this language can read inputs, manipulate data values and produce output as do other high-level languages. The unique feature of this language, however, is that its primary purpose is to generate highly specific FORTRAN code based upon user requests. The FCF "program" can "read" the Input File to determine which sections of FORTRAN code are to be included, what the values of certain constants are to be and/or what analytic equations are to be inserted in to FORTRAN statements.

A number of language statements can be grouped into logical units, referred to as rules. Rules can be thought of as either a main program (the main rule) or a subroutine (a rule invoked by another rule). Rules are logical units in their own right and FLTOPS allows the capability of satisfying rules from sources external to the FCF.

The most basic capability required of the precompiler is the conditional or unconditional inclusion of a block of FORTRAN code. To provide this, a number of FCF control statements with programming-language-like syntax are available. In the case of the Input File example on Angle of Attack Corrections, the following FCF code may be used:

```
P     IF FIND(ANGLE_OF_ATTACK) THEN
P        DO
P        FIND(@.CORRECTIONS)
P        INPUT(WORD)
P        IF * <> 'NONE' THEN
P           DO
P           IF FIND(@.BOOM_BENDING) THEN INVOKE BOOM_BENDING
P           IF FIND(@.DYNAMIC_LAG) THEN INVOKE DYNAMIC_LAG
P           IF FIND(@.PITCH_RATE) THEN INVOKE PITCH_RATE
P           END
P        END
```

Note, in this example, if the node ANGLE_OF_ATTACK is not found on the Input File or the subnode CORRECTIONS is followed by NONE, then none of the FORTRAN code is included. Otherwise the type of correction is determined and that particular code is generated by the BOOM_BENDING, DYNAMIC_LAG and PITCH_RATE rules.

Also note that because the FORTRAN statements can be intermixed with the FCF language statements, FCF language statements are distinguished from the FORTRAN statements by the presence of a "P" in the first character of an FCF line. FORTRAN statements are immediately written to the output file and FCF statements are "executed" as they are encountered. The rule BOOM_BENDING may look like this,

```
P  BOOM_BENDING:
P     SAVE_POINTER
P
C****************************************************
C      BOOM_BENDING CORRECTIONS ARE REQUIRED
C****************************************************
P
P  /* GET THE CONSTANTS TO BE PUT INTO SUBROUTINE BOOM */
P     DEFINE BOOM_CONSTANTS(50)
P     SET(I = 0)                 /* LET I COUNT */
P     DO FOR EACH SUBNODE OF @
P        INCREMENT(I)
P        INPUT(WORD)
P        ENTER(BOOM_CONSTANTS(I+1),'DATA ',*,'/')
P        INPUT('=',CONSTANT)
P        CATENATE(@,'/',';')
P     END
P     SET(BOOM_CONSTANTS(1) = I) /* LET (1) BE TOTAL */
         .
         .
         .
P     RESTORE_POINTER ;
```

The FCF can be arbitrarily complex by using the concept of the FCF rule and the other various language statements available (see the following chart, "Some Legal FCF Commands"). The "Conditional" and "Block/Loop" statements are self explanitory. The "Symbol Table Operations" (DEFINE, SET, INCREMENT, ENTER and CATENATE) allow the intermediate storage and manipulation of character and numeric data essential in the building of FORTRAN statements. "Input File Operations" (SAVE_POINTER, INPUT and RESTORE_POINTER) allow the input of external control data from the Input file supplied by the user. The "Output Operations" statements provide the capability of printing a customized FORTRAN statement to the output file or of sending an informative message to the error file. "Built-In Functions" add additional string manipulation capabilities and the "Debug functions" commands are tools to assist the programmer in providing error free code. (Note that the capability to unconditionally jump (GO TO) has not been designed into the language.)

Some Legal FCF Commands

```
Conditional
     IF <bool exp> THEN <stmt>
     IF <bool exp> THEN <stmt> ELSE <stmt>

Block Loop
     DO <block of stmts> END
     DO WHILE <bool exp> <block of stmts> END
     INVOKE <rule>

Symbol Table Operations
     DEFINE <table>
     ENTER <table>, <symbol>, <value>
     CATENATE <table>, <symbol>, <value>
     REPLACE <table>, <symbol>, <value>
     DELETE <table>. <symbol>
     CLEAR_TABLE <table>
     SYMBOL_EXISTS <table>, <symbol>
     TABLE_EMPTY <table>
     SET <symbol> = <value>
     INCREMENT <symbol>
     DECREMENT <symbol>
     DO FOR SYMBOL TABLE <table> <block of stmts> END

Input File Operations
     INPUT <input element>
     FIND <tree>
     DO FOR EACH SUBNODE OF <tree> <block of stmts> END
     NODE_EXISTS <tree>
     SAVE_POINTER
     RESTORE_POINTER

Output Operations
     OUTPUT <output element>
     MESSAGE <severity code> <message>

Built In Functions
     INDEX <string exp>, <string exp>
     LENGTH <string exp>
     SUBSTRING <string exp>, <start pos>,
     VALUE <table>, <symbol>

Debug Functions
     ASSERT <boolean exp> ERROR: <statement>
     STATUS <string>, <table name>, SYMBOL name>
```

Another feature of the FCF language is "placeholder replacement." This feature allows placeholders to be placed in the FORTRAN code and then replaced just prior to actual output of the code. For example, the FCF statements

```
P    INPUT(NUMBER)
P    SET(#NOPTIONS = *)
        DIMENSICN A(#NOPTIONS), B(#NOPTIONS)
```

cause A and B to be dimensioned to some number specified in the Input File. Placeholder replacement can occur at two different levels, (1) immediate replacement as it is parsed in the code with a dynamically changing value, or (2) replacemnt by a calculated or created value during the second pass through the code. (For example, the first kind is useful if the values of the placeholders are to be read from the input file, like constants, and the second kind is necessary for values not known at the time of the first occurance, like calculated dimension sizes.)

III) USAGE

The intended use of the FLTOPS system was to optimize several of the large existing software systems at [illegible] (SANDS, UFTAS, MMLE, etc...). Complaints from the [illegible] cited complicated options written in computer jargon [illegible] causing several project errors. Also data reduction equations were so far embedded into this multitude of options [illegible] was difficult to understand or change. Project [illegible] complained of wasted storage and execution time [illegible] options which were never used, or never intended [illegible] his particular project. The maintenance of the software [illegible] becoming unmanageable. FLTOPS was designed and [illegible] solve the above problems, but it was also designed [illegible] more general. In fact it's not "Flight Test Oriented" at all.

To begin with it was designed to work on FORTRAN [illegible] regardless of it's orientation (any kind of code [illegible] generated, with some loss of generality, by using the [illegible] mode). It has a small FORTRAN lexical analyzer to perform FORTRAN identifier replacement (if desired) and to warn the user if illegal FORTRAN code is being generated. So any large complicated system experiencing the above problem could benefit from the use of FLTOPS. But it needn't even be a large system.

A system designer could include entire sections of debug and diagnostic testing code during his development stage. This code could be completely removed when the final product is delivered by simply "flipping a switch" at generation time. Or as the case with FLTOPS itself, the code can be designed to be completely portable by writing the machine dependencies into options which can be switched before generating the code for the target machine (see the example on the following page). Subsystems can even be written to be compatible with several systems even though they may require different storage sizes, use different error routines and/or require different data passing linkage.

Other incidental uses include the ability to change all occurrences of an identifier to another (since FLTOPS has it's own mini FORTRAN lexical analyzer, it is smarter than most editors and the A in A = 5 is distiguished from the A's in DATA). It can also change all hollerith strings to quote delimited strings (or apostrophes or any other delimiter desired). One only needs to gain a little experience before understanding the power of the tool.

```
P ADDCH:
      SUBROUTINE ADDCH (ISTRING,IADCH)
P   INVOKE VERSION
C
C   ADD A LEFT JUSTIFIED CHARICTER (IADCH) TO A STRING (ISTRING)
C
C   ***** VERY MACHINE DEPENDENT ******
      DIMENSION ISTRING(#VS256)
P    IF SYMBOL_EXISTS(MACHINE,PRIME) THEN
P       DO
      DATA MSK/:177400/
      NXWORD = RS(ISTRING(1),1)
        .
        .
        .
P       END
P       ELSE IF SYMBOL_EXISTS(MACHINE,CDC) THEN
P       DO
      NXWORD = ISTRING(1)/10
        .
        .
        .
       END
     .
     .
     .
```

IV) SUMMARY

FLTOPS is a powerful software development tool. It was developed to reduce the time and expense of creating specialized software for each new flight test project at AFFTC. It features a readable hierarchical input file and a high-level structured programming language to read the input specifications and generate the appropriate source code. The input can be in terms which relate to the [illegible] instead of in a form only programmers [illegible] FORTRAN-like [illegible] language [illegible] powerful [illegible] extensive [illegible] options and simple [illegible] The use of FLTOPS as a development tool [illegible] [illegible] users.

For further information about FLTOPS, contact [illegible] Baker or Larry Gordon by calling [illegible] 277-4[illegible]74 [illegible] or writing to them at

Air Force Flight Test Center
6520th Test Group/ENCS stop 200
Edwards AFB, California 93523

DISTRIBUTION LIST

	COPIES
EXECUTIVE COMMITTEE	
Mr. W. R. Braswell (KMR)	1
Dr. R. H. Duncan (USAWSMR)	1
Mr. J. L. Wymer (USAYPG)	-
Col R. A. Gardner, USAF (ESMC)	1
Mr. A. D. Phillips (AFFTC)	1
Mr. A. L. Freeman (AD)	1
Col M. L. Bishop, USAF (AFSCF)	1
Mr. K. George (WSMC)	1
Col J. W. La Casse, USAF (TFWC)	1
Mr. G. R. Schiefer (NWC)	1
Mr. W. L. Miller (PMTC)	1
Mr. T. Ramirez (AFWTF)	1
Mr. [illegible] (NATC)	1
[illegible] REPRESENTATIVES	
[illegible] Maxey (KMR)	1
[illegible] Vick (USAWSMR)	1
Mr. [illegible] Steele (USAYPG)	1
Mr. M. [illegible] Real, Jr. (WSMC)	1
[illegible] V. Lochman (ESMC)	1
Mr. A. F. Miller (AFFTC)	1
Mr. J. F. McLain (AD)	1
Mr. R. A. McLellan (TFWC)	1
Mr. R. V. Boyd (NWC)	1
Mr. W. R. Hattabaugh (PMTC)	1
[illegible] R. Cotter, USAF (AFSCF)	1
[illegible] J. Brown (AFWTF)	1
[illegible] P. M. Davis (NATC)	1
[illegible] CHAIRMAN	
[illegible] Hackamack (USAYPG)	2
[illegible] VICE CHAIRMAN	
Mr. J. P. Welch (AD)	2
[illegible] MEMBERS	
Mr. A. Baswell, Jr. (KMR)	1
Mr. F. J. Bailey, III (ESMC)	1
Mr. W. H. Nolin (ESMC)	1
Mr. R. N. Barry (AFFTC)	1
Mr. R. Taketa (AFFTC)	1
Mr. J. Papa (AFFTC)	1
Mr. [illegible] Fletcher (WSMC)	1
Maj [illegible], USAF (TFWC)	1

	COPIES
Mr. J. B. Gush (TFWC)	1
Mr. [illegible] Manion (NWC)	1
Mr. T. E. [illegible] (PMTC)	1
Mr. [illegible] Greenwald (PMTC)	1
[illegible] E. H. [illegible] (USAWSMR)	1
LtCol [illegible] Hi[illegible], USAF (AF[illegible])	1
Mr. [illegible] Rodriguez (AFWTF)	1
Mr. [illegible] Johnson (NATC)	1
Mr. [illegible] Herndon (NATC)	1

[illegible]

[illegible]	
[illegible] (FLTA[illegible]	1
[illegible]	
Mr. [illegible] R. [illegible] (NASA-MSFC)	1
[illegible] (NASA)	1
Mr. R. R. [illegible] (DOT/FAATC)	
[illegible], USAF (ADWC)	

GROUP CHAIRMEN/VICE CHAIRMEN

Mr. F. D. Roth (PMTC), DG Chairman	1
Mr. W. Kupferer (AFFTC), DG Vice Chairman	1
Mr. W. C. Meeks (USAWSMR), RSG Chairman	1
Mr. R. W. Pogge (NWC), RSG Vice Chairman	1
Mr. R. Lane (WSMC), JRIAIG Chairman	1
Mr A. Borrego (USAWSMR), ETMG Chairman	1
Mr. S. G. Fields (NWC), ETMG Vice Chairman	1
Mr. G. W. Miller, Jr. (USAWSMR), FMG Chairman	1
Mr. E. Rowe (AFFTC), FMG Vice Chairman	1
Mr. E. J. Keppel (AD), MG Chairman	1
LtCol J. W. Oliver, USAF (ESMC), MG Vice Chairman	1
Mr. L. G. Reese (AFFTC), OSG Chairman	1
Mr. M. Skinner (USAYPG), OSG Vice Chairman	1
Mr. R. A. Stimets (USAWSMR), TCG Chairman	1
Mr. D. Hachadorian (USAYPG), TCG Vice Chairman	1
Mr. C. G. Ashley (PMTC), TG Chairman	1
Mr. D. K. Manoa (WSMC), TG Vice Chairman	1
Mr. G. A. Nussear (PMTC), USG Chairman	1
Mr. C. Ramos (AFWTF), USG Vice Chairman	1
Col L. Wolfe, USAF (WSMC) IOAHG Chairman	1
LtCol R. W. Large, USAF (AFFTC), IOAHG Vice Chairman	1

COPIES

Ms. G. E. Hunt
Associate Head, Systems Technology Dept.
Code 1201, Building 512
Pacific Missile Test Center
Point Mugu, CA 93042 1

Mr. W. L. Egan
Head, Systems Engineering Branch
Code 1221
Pacific Missile Test Center
Point Mugu, CA 93042 1

Mr. T. R. Berard
6520th Test Group/ENCS
Air Force Flight Test Center
Edwards AFB, CA 93523 1

END

11-86

DTIC